Naval Ocean Systems Center
San Diego, CA 92152-5000

Technical Document 2171
September 1991

Development of a Modular Robotic Architecture

R. T. Laird
R. P. Smurlo
S. R. Timmer

NAVAL OCEAN SYSTEMS CENTER

San Diego, California 92152-5000

J. D. FONTANA, CAPT, USN
Commander

R. T. SHEARER, Acting
Technical Director

ADMINISTRATIVE INFORMATION

The work in this report was performed during FY 91 by the Advanced Technology Branch (Code 535) of the Naval Ocean Systems Center as a project of Independent Exploratory Development (IED) program. Sponsorship was provided the Office of Chief of Naval Research, Arlington, VA.

Released by
S. W. Martin, Head
Advanced Technology
Development Branch

Under authority of
D. W. Murphy, Head
Advanced Systems Division

PK

CONTENTS

APPENDICES

FIGURES

1.0 INTRODUCTION

1.1 OBJECTIVE

The objective of this project is to develop the hardware and software components for constructing and controlling reconfigurable, modular robots (MODBOTs). While numerous robotic control architectures currently exist, few offer the degree of flexibility and modularity that is required to support rapid integration and prototyping of evolving (processor and sensor) technology into demonstrable systems. The proposed architecture emphasizes standard electrical and mechanical hardware interfaces between distributed processing modules, and standard software libraries that provide communication services and process control across a wide range of processors. The product is a standardized set of tools for building robotic systems that can be easily reconfigured as project requirements and technology change.

The first-year effort is concentrating on specification and design of the architecture, while the second- and (potential) third-year efforts will pursue implementation and demonstration of the MODBOT concept as applied to an actual application, such as physical indoor security.

1.2 SCOPE

This document describes the high-level architecture requirements and introduces the concepts related to MODBOT systems. The preliminary design and an example application of the architecture are detailed. The material is divided into the following sections:

Section 1.0 is an overview of the modular architecture.

Section 2.0 discusses the reasons why a new robotic architecture is needed, and gives examples of various MODBOT applications (both immediate and future). The section includes an overview of previous work, and briefly summarizes related systems that were investigated while developing the modular architecture.

Section 3.0 presents the requirements of a MODBOT architecture, and outlines what it must support in terms of capabilities, from both the developer's and the user's point of view.

Section 4.0 describes the MODBOT system architecture in terms of the major hardware and software subsystems. Details on the application of the architecture to a mobile security robot are included.

Section 5.0 describes MODBOT operation in terms of automatic or built-in capabilities such as self-diagnostics, self-configuration, and self-preservation. The communication and the coordination of multiple robots are outlined.

Section 6.0 presents a brief system development plan. Independent development of the MODBOT sensor, the actuator, and the processor modules is also discussed.

Section 7.0 defines the various acronyms and abbreviations used throughout the text.

Section 8.0 lists the reference material.

1.3 OVERVIEW

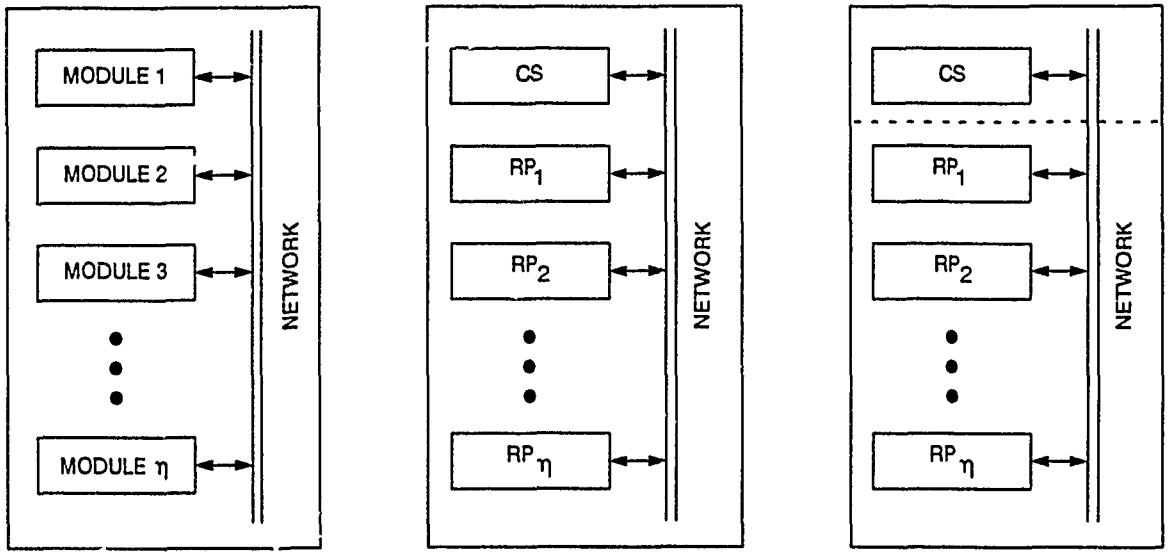
The Modular Robot Architecture (MRA) describes both the hardware and software components that are used to create a MODBOT. The MODBOT itself is a generic entity that must be customized by the developer or intelligent user for a particular application. The MRA facilitates customization of the MODBOT for specific tasks by providing sensor, actuator, and processing modules that can be configured in the manner demanded by the application. The Mobile Security Robot (MOSER) is an example of a MODBOT that will be developed using the modular architecture.

Conceptually, the MODBOT is similar to the IBM PC with its expansion slots; adding a module to a MODBOT is like adding a peripheral card to a PC. One simply plugs a card into an available slot, installs the supplied software drivers, and immediately incorporates the new capabilities of the card into the system. Adding smarter, better, and faster modules and capabilities to a MODBOT will be equally simple. The ability of the MODBOT to accept modules of increasing complexity provides the MRA with its evolutionary growth potential, and has been a primary motivating factor for the development of the architecture.

Simply stated, a MODBOT is a collection of independent modules of varying intelligence and sophistication connected together by a generalized, distributed network. The MRA does not require a particular physical module configuration nor does it require that all modules be located physically together. The generic MODBOT is illustrated in figure 1.

For systems involving direct human supervision, a MODBOT is divided into two physically separate computing systems: the Control Station (CS) and the Remote Platform (RP). The CS is a single module that is remote from the rest of the MODBOT. The RP consists of several modules and is connected to the CS by a telemetry link that acts as a network bridge. Two possible implementations to this approach are given in figures 1b and 1c. Only in systems that are strictly autonomous would the CS be located with the RP (figure 1b).

(Since most of the systems developed using the MRA will involve remote human control at various levels, the division of the MODBOT into separate CS and RP systems will be assumed throughout the remainder of this document.)



(a) generic MODBOT

(b) control-station module located with remote platform

(c) control-station module separate from remote platform.

Figure 1. Robot module configurations.

The MRA is the framework around which robotic applications can be developed. The MRA supplies a set of standard hardware and software components that the user then assembles to build a modular system that can be easily upgraded as requirements and technology change. Standardized components provide a common interface for integrating the various parts of a system such as sensors, processors, and information. From a developmental viewpoint, the MRA is the "glue" that holds the pieces together.

A variety of applications can be addressed with the modular architecture, from indoor security to outdoor surveillance, but the architecture is especially useful for developmental or prototype applications. (Each hardware implementation of the architecture addresses a different set of robotic applications.) The ability to reconfigure and change modules lets developers quickly test new technology with a minimal amount of integration overhead (and expense). Figure 2 shows the relationship between the components of a typical modular system in an indoor application.

Modular robots will be particularly valuable in the laboratory environment where requirements continually change. Ideas can be implemented and tested quickly in a modular fashion and, when satisfactorily debugged, transferred to the deliverable system.

Operation of a MODBOT depends primarily upon the application. The MRA provides a software "kernel" around which application-specific control methodologies and algorithms can be implemented. Only rudimentary process control is provided by the modular architecture. More sophisticated coordination must be supplied by the developer as required.

The MRA is a generic tool that must be customized from both a hardware and software standpoint before a particular application can be addressed. The flexibility of the MRA, made possible by standard interfaces and a variety of hardware and software "hooks," allows developers to configure a robot to specific needs. Standardized interfaces and a modular hardware design allow for independent development of sensor, actuator, and processor subsystems that can be tested and debugged offline. Final system integration is performed much more efficiently since the functionality of the components being added has already been verified.

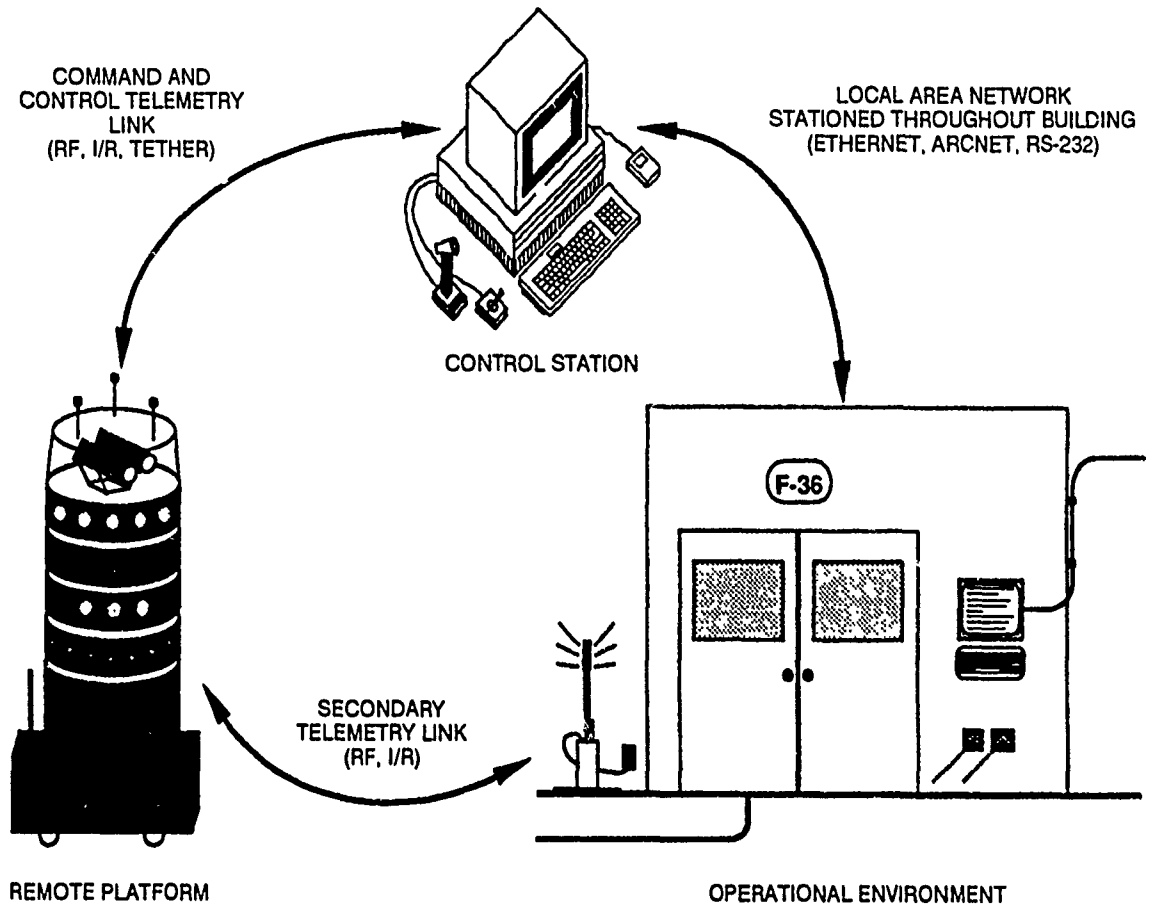


Figure 2. The relationship between components of a modular-robotic system.

2.0 BACKGROUND

2.1 THE NEED FOR A MODULAR ARCHITECTURE

Most current and projected nonindustrial Navy applications of robotics involve mobile systems. The development of a suitable processing and control architecture is a task that historically has been independently undertaken for individual robotic projects, each addressing different operational needs. Results often do not succeed due to insufficient funding, awareness of the issues and alternatives, or a tendency to become outdated. Furthermore, the majority of efforts have produced application-specific control systems that are difficult to adapt to more than the problem at hand. The development of a flexible, powerful, and widely available "core" high-level processing system with evolutionary growth potential for use on mobile robots (ground, air, surface, and underwater) will greatly alleviate these problems. An atmosphere of standardization and compatibility can be fostered among systems throughout the fleet.

A standardized, modular control system will also reduce the costs associated with development of customized architectures. Only the configuration of the pieces would have to be done each time, not the redesign of the entire system. In addition, a standardized architecture will promote software/hardware reusability in that identical modules and components will be used in several places, reducing both development time and cost.

The MRA is a generic control system with a standard set of hardware and software tools that can be used to design modular robots with a high degree of flexibility and extensibility. Several architectures currently exist that can be used to construct the control mechanisms for complex (robotic) systems. The Realtime Control System (RCS) developed by the National Bureau of Standards (NBS, also known as National Institute of Standards and Technology [NIST]), is an example (Barbera, Fitzgerald, & Albus, 1982). The MRA, however, is designed specifically to support the development of modular robots and modular control systems (in the general case). The MRA emphasizes standard hardware (electrical/mechanical) and software interfaces to promote development of capabilities by multiple activities (e.g., Navy, Army, Air Force, Marine Corps), the products of which can then be easily integrated to form a cooperative solution to a common problem.

2.2 APPLICATIONS OF THE MRA

The MRA can be used on a variety of applications ranging from a simple embedded device control to sophisticated autonomous robot control. Very little in the *specification* of the architecture restricts its use to a given class of control applications. Typically, however, a particular *implementation* will restrict the architecture to a specific

set of problems. The implementation described in this document is aimed at the control of mobile robots.

2.2.1 Modular Developmental Testbed

The primary purpose of the MRA is to support the development of robot modules and control algorithms that will be used to build MODBOTs. Module development and integration is facilitated by the use of standard interfaces and procedures. Developers are required to conform to the standards, but are allowed a great deal of flexibility in the types of modules that can be developed. The actual control methodology (i.e., how the robot's action is controlled) is also "modular" in a sense and can be modified by the developer to obtain any desired behavior.

The module concept allows for independent development of new sensor, actuator, and software control components. These components are typically developed as modules for MODBOT applications, but the modules may serve as a simple means for testing new components that are not necessarily destined for MODBOTs (or even robotic applications).

Once the hardware modules and control algorithms have been developed, specific applications can be addressed. (Useful systems must solve real-world problems, and the architectures upon which they are based must be proven capable of performing. The developmental testbed is necessary but, alone, is not sufficient to solve problems.)

2.2.2 Physical Security

The first instance of a MODBOT to be developed under the MRA will be the Mobile Security Robot (MOSER). MOSER addresses the need for physical security within the confines of a structure such as an office building or a warehouse. The security robot will be an autonomous, modular, mobile system responsible for detecting intruders and responding to the assessed threat. It will duplicate several of the sensor and processing components found on ROBART II (Everett et al., 1990) with several improvements in the area of distributed processing, such as system networking, modularity (module design), and dynamic system configuration.

This application was chosen as the first implementation of the MRA for a number of reasons: (1) the familiarity of the security task to the development team, (2) the current availability of an existing, successful security robotic testbed (i.e., ROBART II), and (3) the desire to develop an inhouse mobile (security) robot capability.

MOSER is intended to be a fully autonomous system capable of operating within its environment without human supervision. However, because intelligent autonomous control is not readily achieved, MOSER's capabilities will be extended incrementally as

levels of control (from teleoperated to autonomous control) are added to the robot's control scheme.

MOSER consists of the following hardware systems (items 4 to 7 are not specifically discussed here):

1) Control Station (figure 2).
2) Remote Platform (figure 3).
3) Telemetry Link (initially a tethered cable).
4) Environmental processor.
5) Fixed environmental sensors and actuators.
6) Navigational aids (i.e., beacons, freeway markers)
7) Local Area Network stationed throughout environment.

The software portions of the MRA will be used to tie the hardware systems together through the standard interfaces mentioned above. This document simply introduces MOSER and physical security as an application of the MRA; detailed design information for MOSER would be given in hardware and software design specifications.

2.2.3 Other Applications

Development of the MRA gives the Navy a rapid-robotic-prototyping capability, and makes immediately available the tools to construct modular (robotic) solutions to other problems. Almost any application involving distributed processing on an intermediate scale (e.g., less than 255 processors) can be implemented in a modular fashion. Three possible uses are given below:

Waterside Security

Physical Security could be performed by an outdoor version of MOSER. Problems that arise when moving from indoor to outdoor systems include autonomous navigation (waterside environments being a bit more harsh and unpredictable than an office building or even an enclosed warehouse). A MODBOT could be constructed to patrol a "secure" pier in an autonomous mode (with the aid of fixed beacons, perhaps), and could alert a remote operator of intruders or of the absence of important cargo. The advantages of robotic security guards are numerous (Everett, 1988).

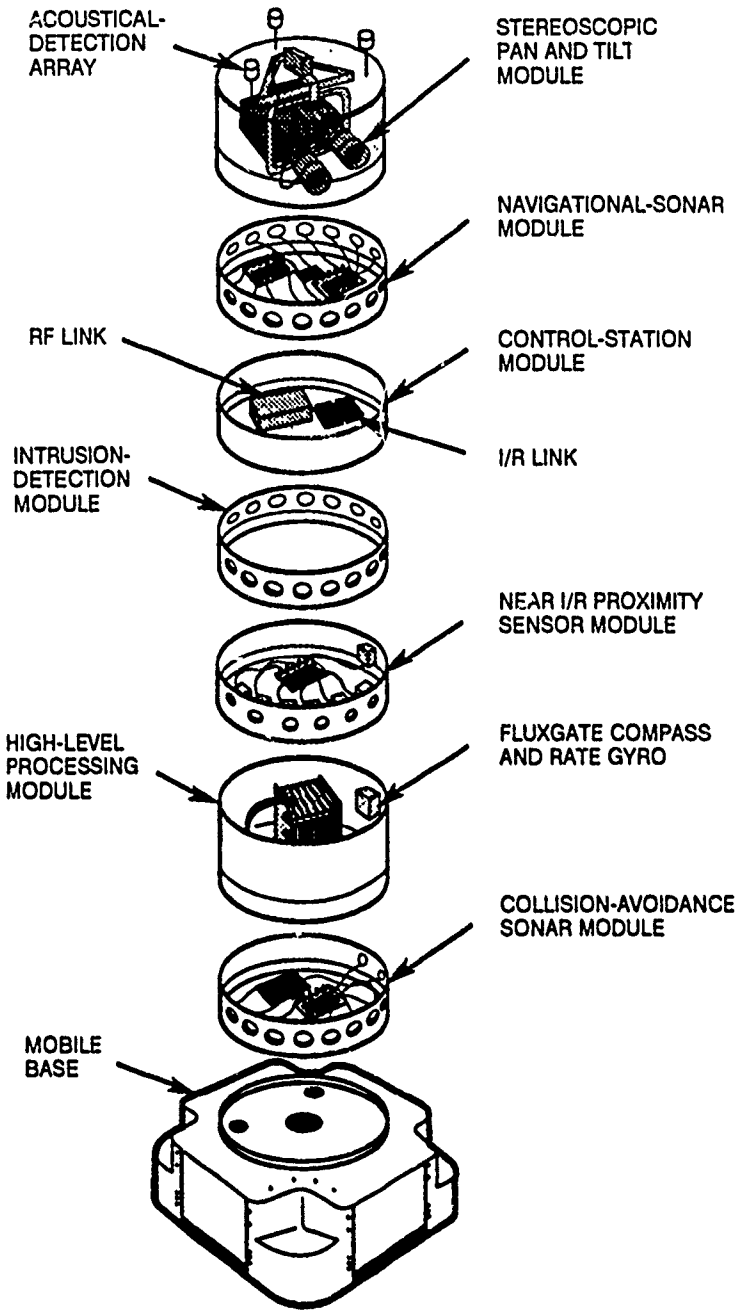


Figure 3. MOSER remote-platform (RP) module configuration.

Underwater Exploration

Either a tethered or an autonomous MODBOT could be used for ocean exploration and surveillance. A MODBOT could be easily adapted to a submergible platform. Special underwater sensors could detect and classify underwater objects. Acoustical (sonar)-sensor modules could also be developed for monitoring and locating underwater

activity to be investigated autonomously or under the supervision of a remote operator. The navigational problem is even more difficult when another dimension is added.

Intelligent Underwater/Surface Sensor

Stationary MODBOTs placed at critical locations near harbors, sea access lanes, or other points of interest could be used as intelligent sensor platforms. Several highly sophisticated sensor modules employed on a single MODBOT would detect the presence (or absence) of specific objects (environmental conditions). The MODBOT could then be programmed to respond by simply recording the event or perhaps respond in a more active manner. In this application, the MODBOT is nothing more than a data-fusion machine with some heuristic applied to generate the desired response.

2.3 RELATED WORK

Extensive research in mobile-platform development has taken place at Stanford, Carnegie-Mellon, MIT, DARPA, Martin-Marietta, NBS, and NOSC (to name a few). The sections below summarize the architectures relevant to this effort that have been researched. Illustrations of the architectures are included for easy comparison between several different approaches to the control of intelligent machines (figures 4 through 9).

2.3.1 Generic Robotic Processing Architecture (GRPA)

The predecessor to the MRA (temporally, if not logically), the Generic Robotic Processing Architecture (GRPA), was used on the Unmanned Ground Vehicle/ Teleoperated Vehicle (UGV/TOV, formerly GATERS) program (Hughes et al., 1990). As implemented on the GATERS project, GRPA was basically a mechanism for mapping operator input on the control station to vehicle actuators on the remote platform, successfully demonstrating a relatively sophisticated degree of teleoperated control. This version of GRPA *implemented* only a portion of the architectural concepts as initially outlined (Aviles, Laird, & Myers, 1988), but did prove that the basic system design was capable of realtime device control (incentive enough to pursue further development of the modular architecture concept).

The GRPA processing architecture (figure 4) is based upon the Virtual Systems Interface (VSI) data structure. Objects (e.g., sensors, actuators, state variables) are divided into four categories: remote-system sensors, remote-system actuators, control-station sensors, and control-station actuators. The control portion of GRPA is responsible for mapping local controls onto remote actuators and for mapping remote sensors onto local displays. Virtual device drivers—hardware-specific I/O routines—are used to couple the physical world to objects in the VSI. A remoting system is used to transmit and receive encoded packets between the control station and the remote vehicle.

Several key features of the MRA are common to the original goals of GRPA (e.g., standard software interfaces, a core high-level control system, and an extendible modular design). The MRA is partially a renovation of some of the GRPA ideas as begun under the Distributed Robotic Control Architecture IED project in FY 88. Whereas GRPA emphasized "compilation" of robot descriptions into actual implementations, the MRA emphasizes distribution of function into modules that can be dynamically configured. GRPA was concerned primarily with the standard software interfaces; the MRA attempts to standardize both the software and hardware interfaces between distributed components. Additionally, system networking concepts introduced in GRPA will be expanded upon and implemented under the MRA.

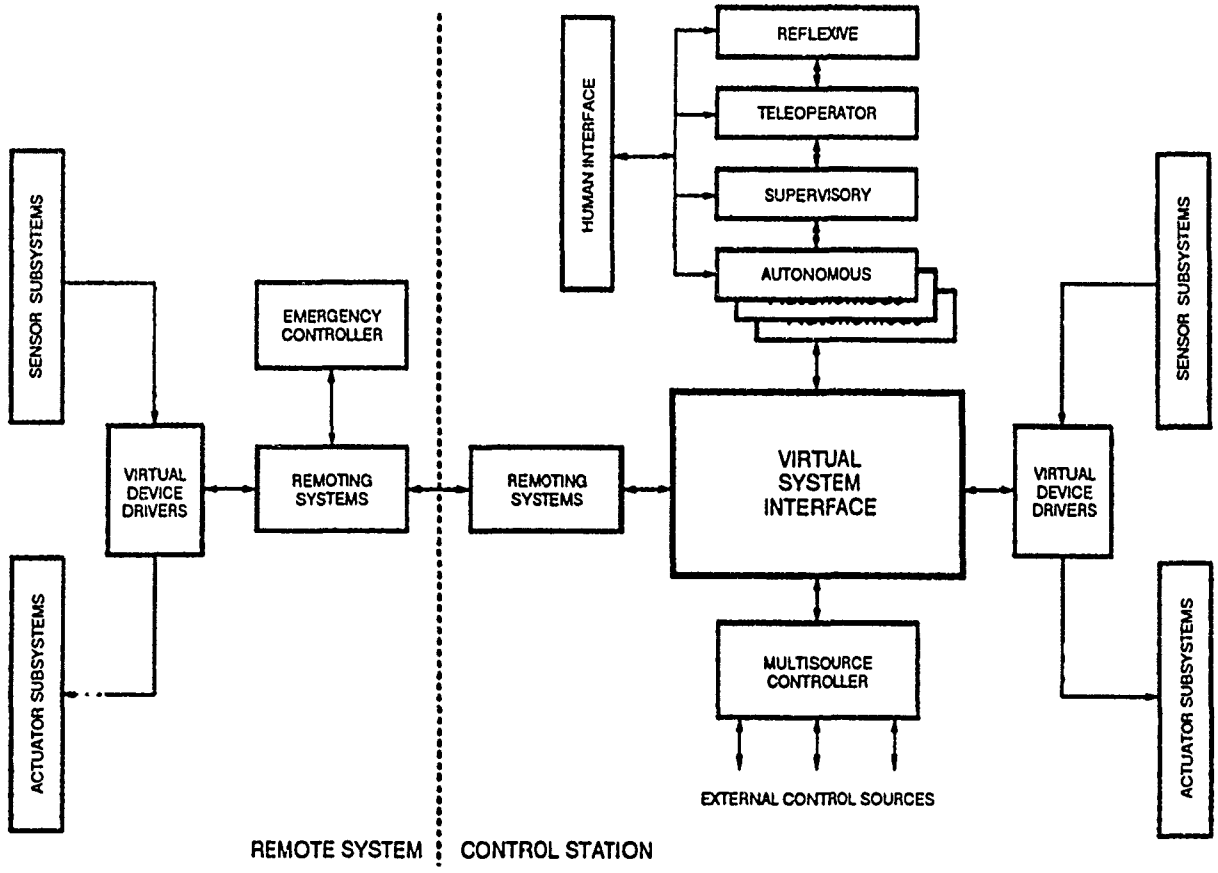


Figure 4. Overall GRPA architecture for all levels (Aviles, Laird, & Myers, 1988.)

2.3.2 ROBART II

ROBART II, an autonomous security robot being used at NOSC, is a testbed for the development of hardware/software solutions to problems facing mobile (sentry) robots. Over the last three years, ROBART II's sentry capabilities have been enhanced to make it one of the most advanced autonomous security robots used by the Navy (Everett et al., 1990).

The computer architecture used on ROBART II (figure 5) is designed as a distributed hierarchy of ten onboard microprocessors acting as dedicated controllers with a remote microcomputer used as the high-level system Planner. The Planner performs the functions of path planning, obstacle avoidance, position estimation, map making, sonar-range plotting, and security assessment. The onboard computers are dedicated to such functions as head positioning, sonar ranging, platform mobility, and speech synthesis; one of these processors acts as the system Scheduler and coordinates the activity of the others. Communication between the onboard computers is controlled by the Scheduler and is accomplished by means of an 8-bit parallel bus and a multiplexed RS-232 serial port. Communication between the remote Planner and the robot is via a 1200-baud radio link.

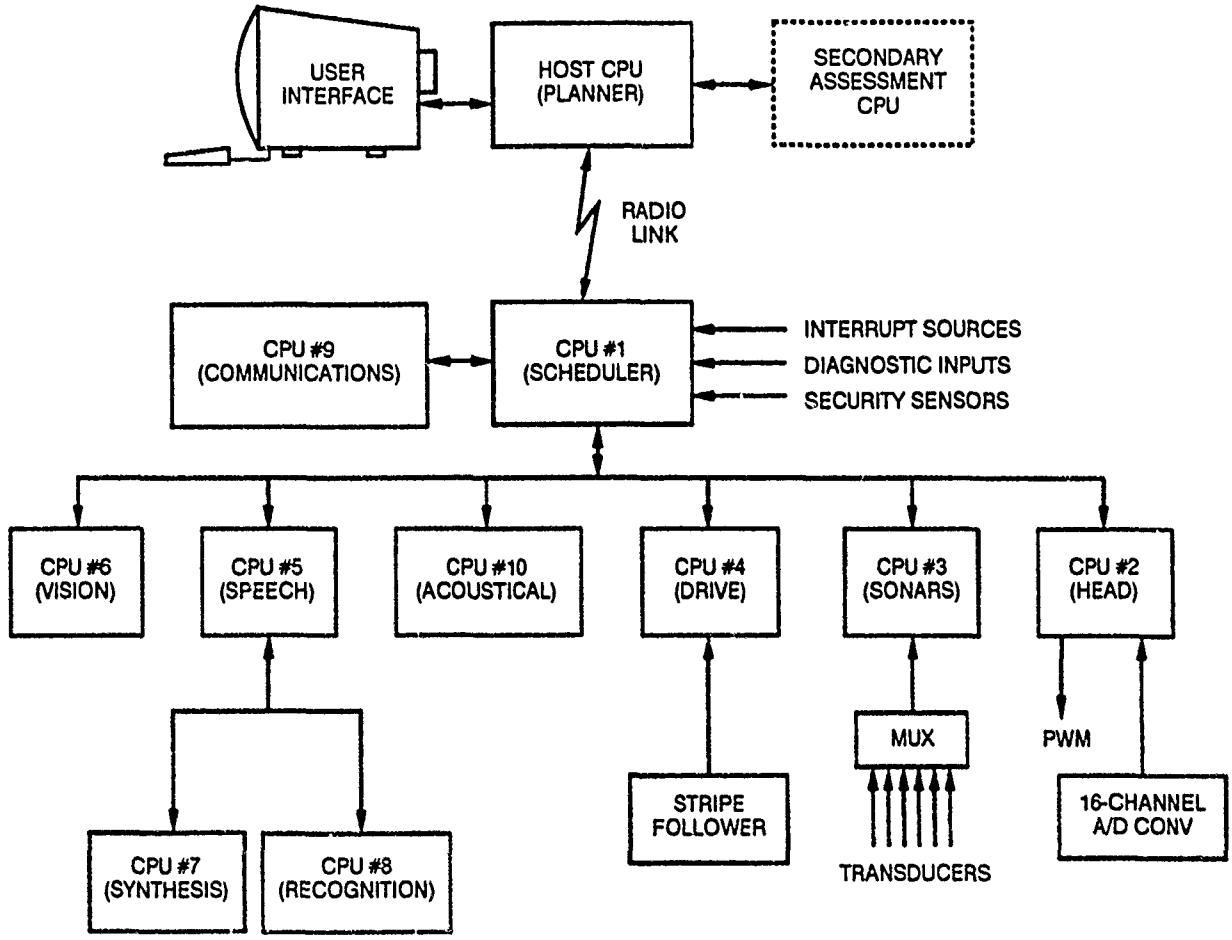


Figure 5. Distributed computer architecture for ROBART II (Everett et al., 1990.)

Because ROBART II is not owned by NOSC, a replacement robot is needed to enable research and development to continue. The MOSER will provide NOSC with a security robot having capabilities on the order of ROBART II as well as the potential to exceed those capabilities.

The MRA will borrow from the experience gained in the design and implementation of ROBART II and offer new solutions to problems that face all evolving, distributed computer systems (e.g., management of growth and effective communication between processing components). Much of the technology used on ROBART II will be transferred to MOSER, particularly the autonomous navigation and security-assessment concepts. This technology transfer will allow the MRA physical-security application to be developed much faster than would otherwise be possible (section 2.2.2).

MOSER's development is *partially* the result of the desire to create a more powerful ROBART II with the ability to easily add new capabilities; integration of additional sensor and processing components is becoming difficult for the developers of ROBART II because the robot's enclosure is nearly full. The modular approach will allow the technology originally intended for ROBART II to be implemented on MOSER (e.g., line following to aid in vehicle navigation).

2.3.3 Other Architectures

There are perhaps hundreds of control architectures that have been developed for use on intelligent robotic systems. However, the literature search revealed four approaches that stand out in terms of both relevance to development of the MRA and in terms of the unique architectural concepts they describe. These efforts influenced the generation of the high-level requirements for the MRA. The paragraphs below *briefly* summarize each of these architectures and provide insight into some of the design decisions made in developing the MRA.

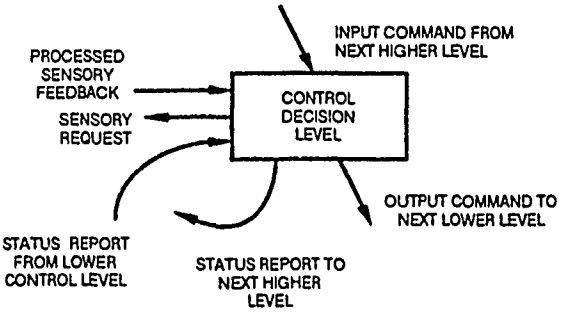
Realtime Control System (RCS)

The RCS, developed by the National Bureau of Standards (NBS) as an environment for the design and implementation of distributed, task-based control applications (Barbera et al., 1984), has been successfully implemented in such efforts as the NBS Automated Manufacturing Research Facility (AMRF), a multirobot manufacturing shop (McGain, 1985).

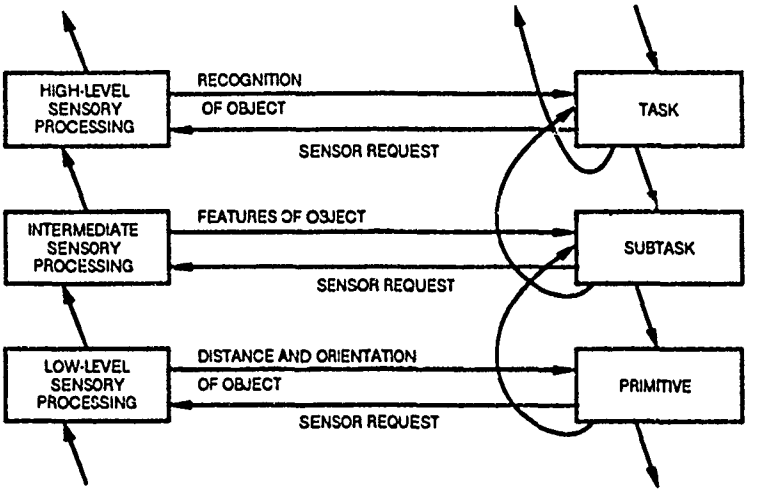
The RCS does not include specifications for the actual control algorithms, but does include methods for organizing and interfacing those algorithms. The MRA is similar in that it specifies only module-level communication and data abstraction; it does not specify the algorithms used to control processes within a module. Both the RCS and the MRA provide developers with a set of tools to implement a variety of control systems in a structured, standardized manner.

The RCS is based on the concept of generic control levels that are organized in a hierarchy based upon a task decomposition of the problem (figures 6a and 6b). Each of the control levels has a standard input and output data-set interface that facilitates modularity. Processing at each control level consists of sampling input data and

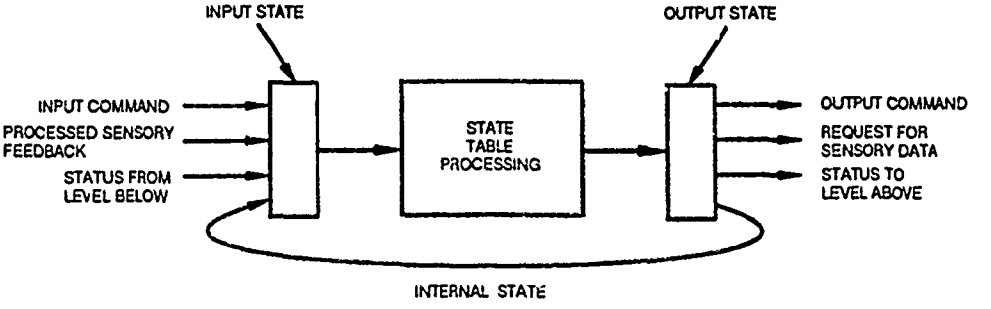
generating output data based upon user-defined state table information (figure 6c). Sensory feedback is provided to each control level by sensory-processing modules available at that level (figure 6b). Realtime response is obtained by distribution of control levels onto multiple processors communicating through common memory.



(a) generic control level



(b) hierarchical structure and sensory-control interaction



(c) state-table inputs and outputs.

Figure 6. RCS architecture components (Barbera, Albus, & Fitzgerald, 1982).

Like the RCS, the MRA defines a set of standard interfaces that allow for modularity and the ability to easily introduce new capabilities. The MRA also uses distribution of function onto multiple processors to obtain realtime response where required. Unlike RCS, however, the MRA does not directly specify a particular control architecture (e.g., hierarchical decomposition); it only specifies how the modules that compose the architecture (whatever it may be) communicate, and what common functionality each module must provide.

NASA/NBS Standard Reference Model (NASREM)

NASREM is a hierarchical control-system architectural model designed by NBS to support development of the control-system architecture for the Flight Telerobot Servicer (Albus, McGain, & Lumina, 1984). NASREM, primarily intended for telerobotic systems, has been extended to include control of autonomous systems as well. As depicted in figure 7, control is divided into three conceptual processes at six different levels. Actions are performed by decomposing higher level commands into tasks that eventually produce real-world results (e.g., manipulator movement). The tasks perform different types of mathematical transformations at each level. An operator interface allows the user to control the system at any one of the six levels. NASREM is an implementation of RCS just as MOSER is an implementation of the MRA. NASREM was based upon the system architecture that evolved from RCS. However, each offers slightly different concepts that the MRA intends to support (or seeks not to exclude).

The goals of the MRA include features offered by and contained in NASREM: flexible and well-integrated system interface, control at various levels of interaction, world modeling at various levels of abstraction, and high-level command execution. Like NASREM, the MRA is a "standard model" that must be customized to meet the requirements of a particular application.

Subsumption

The subsumptive architecture developed by Brooks (1986) is based on task-achieving behaviors organized vertically as horizontal slices (figure 8a). This is different from the traditional horizontal decomposition of tasks (e.g., perception, modeling, planning) into vertical slices. Layers of behaviors can be subsumed by higher levels to produce more intelligent behavior (figure 8b). Layers are composed of modules that are finite state machines with associated data variables (figure 8c).

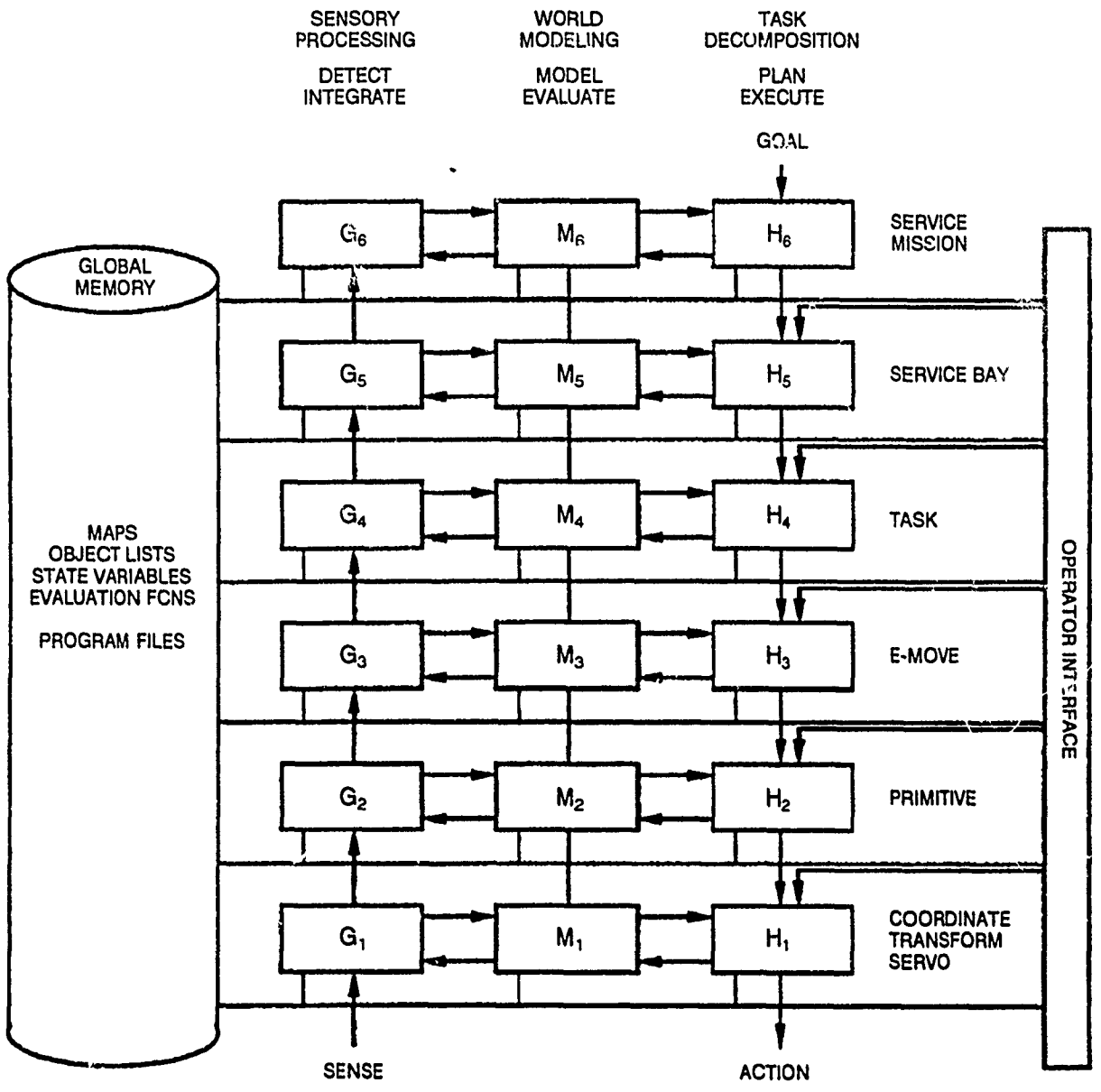
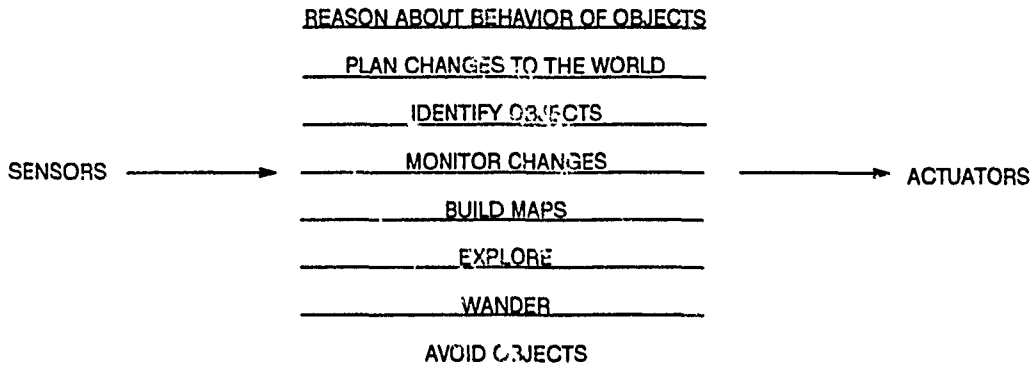
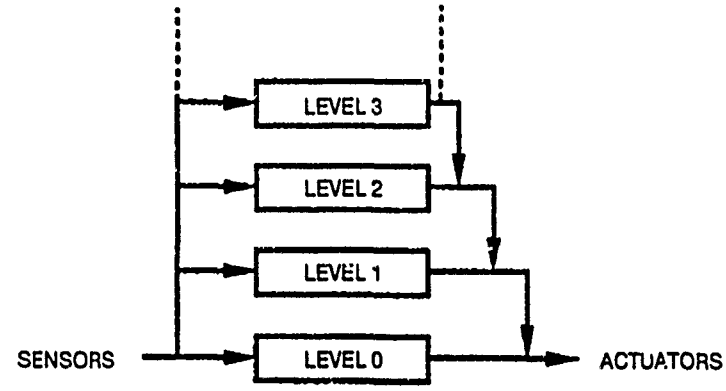


Figure 7. NASREM (RCS) hierarchical control system architecture (Albus, McGain, & Lumina, 1984.)

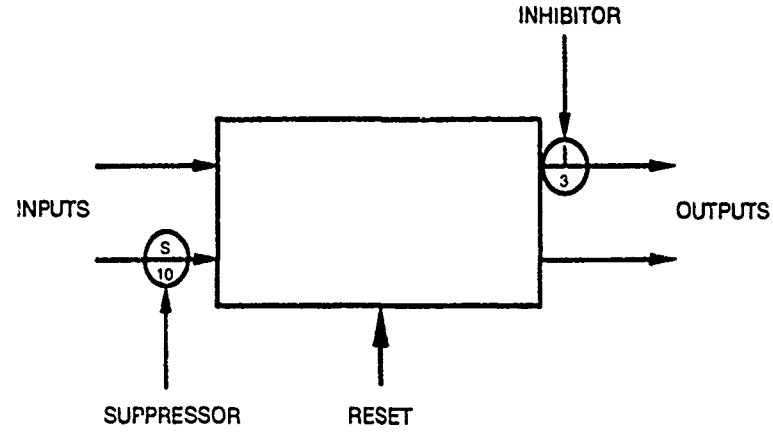
The subsumptive approach is significant in that it provides almost immediate functionality at lower levels of behavior, and allows for incremental growth toward an autonomous, useful purpose. The MRA can support subsumption directly by virtue of its modularity and flexibility in supporting behavior-based control algorithms at the module level. The control system requirements of multiple goals, multiple sensors, robustness, and extensibility identified by Brooks (1986) as being inherent to the subsumptive control architecture, also apply to the MRA.



(a) vertical decomposition of control system into horizontal slices



(b) layered control levels subsumed by higher levels



(c) subsumption module with input and output lines.

Figure 8. Subsumption architecture (Brooks, 1986).

Nested-Hierarchical Controller (NHC)

Figure 9 shows the architecture of the NHC that its developer (Meystel, 1988), states as being an integral part of all autonomous mobile robots (AMR). The NHC offers concepts that are applicable to the development of the MRA and to MODBOT implementations such as MOSER. Of particular interest is the decomposition of motion into planning, navigating, piloting, and control. This can easily be implemented under MRA by using distinct modules to carry out each of the levels of motion. Each module could also implement the corresponding level of perception and knowledge representation used by the motion process. Actuators and sensors can also be modeled and implemented under the MRA in a manner similar to that of the NHC.

The NHC architecture is divided into three functional areas: perception (how the robot sees its environment); knowledge (how the robot models the perceived environment); and planning (how the robot accomplishes goals and reacts to environmental situations). Each of the three areas is broken down into levels of data abstraction that are useful to the corresponding level of control. Direct sensory feedback is available to the lower-level control subsystems for realtime and reflexive response.

Although architectures do exist for building modular robotic systems, the modular approach proposed does not force a particular control scheme upon the developers. Instead, it offers rudimentary control mechanisms that can be replaced by those supplied by system developers. In addition, very few systems offer the degree of hardware modularity and flexibility that is achieved through *simple*, standardized interfaces supported by the modular architecture.

Each of the architectures above can be implemented using the components of the MRA. The modular architecture itself offers only a simple default controller. More sophisticated robot control can be obtained by implementing other methodologies in a modular fashion by using selected (standardized hardware/software) components; in this case, the modular architecture is nothing more than a development and integration tool. The modular architecture is not intended to replace those above, but to facilitate their implementation as modular systems.

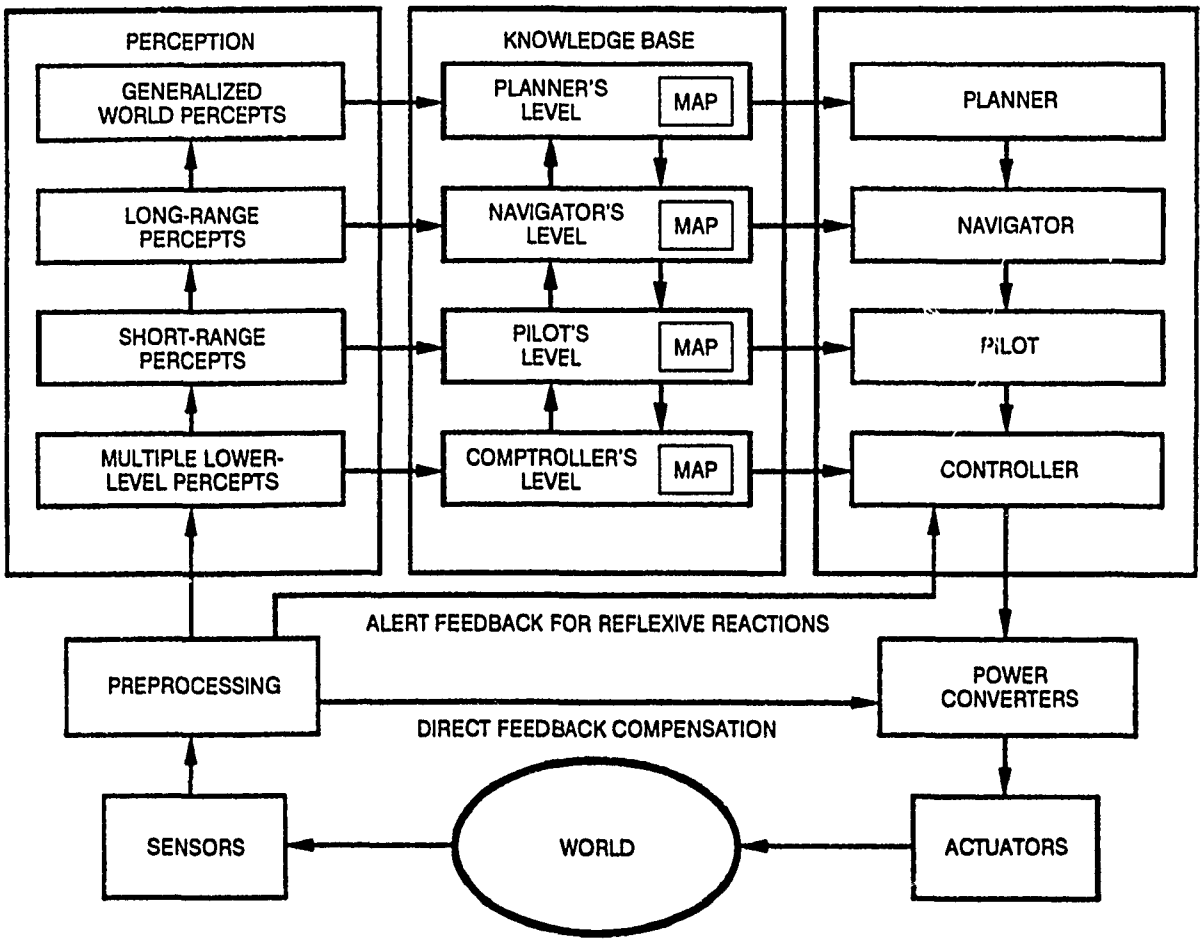


Figure 9. The Nested-Hierarchical Controller (NHC) architecture (Meystel, 1988).

3.0 A FRAMEWORK FOR DEVELOPING MODULAR SYSTEMS

3.1 SPECIFICATION REQUIREMENTS

Requirements, even for systems that are *not application specific*, drive the design and should be specified in advance. These requirements specify the characteristics of an architecture that can be used to create modular systems with a great degree of flexibility in terms of hardware configuration and software operation. If the architecture possesses these characteristics, then it should be capable of supporting construction of modular robots.

The need for a new architecture was discussed in section 2.1. This section outlines the high-level software and hardware requirements. In the sections following, the MRA is referred to as "the architecture".

3.2 MRA HARDWARE REQUIREMENTS

The general hardware requirement is for an architecture that will support production and assembly of sensor, actuator, and processor modules that can be connected together using a generalized power, communications, and auxiliary control/feedback "bus." The MRA provides hardware components and guidelines that system developers use to build modular robots. Specific hardware requirements are listed below.

3.2.1 Hardware Design

Emphasis is placed upon a simple, flexible, and maintainable hardware design. Complex designs lead to complex problems that require complicated tools to solve. These components will have simple, standardized interfaces that facilitate system integration.

3.2.2 Support for Add-On Modules

Incremental development of capabilities will be supported by add-on sensor, actuator, and processor modules that can be configured in any manner required. Modules will typically be constructed from materials that allow easy physical connection, and are structurally adequate for the operating environment. The modules may be separate and connected only by wires that form the generalized module bus. The architecture will support, at a minimum, 16 modules in a single system.

3.2.3 Standard Platform Interface

The architecture will provide a standard platform interface that decouples the rest of the modular robot from the vehicle platform or base. The interface is a mechanical

and electrical connection between the platform and one of the robot modules (for mobile systems this will be the "mobility" module). Standardizing this connection allows a modular robot to be moved between platforms without modifications. The platform interface is also responsible for regulating power supplied by the base to standard voltages that are made available to the rest of the robot. In terms of power, the platform must be capable of supplying 24V DC (raw), which will then be regulated to standard voltages of +12V DC and +24V DC (clean).

3.2.4 Standard Module Interface

Modules are connected to *each other* through a common communications and power bus. The bus provides a standard interface that specifies the required mechanical and electrical connections for modules. The interface should allow for growth in the bus (auxiliary control and feedback lines, for example). The bus, a conceptual device, is implementation dependent. The bus may be a set of wires, a connectorized backplane, or a combination of both. The communications portion of the bus will support rates of at least 1.8 megabits-per-second (Mbps) with the ability to detect and correct for simultaneous (interfering) bus-access requests. The power portion of the bus will supply a minimum of 10A at +12V DC and 15A at +24V DC. Standard voltages of +5V DC (or +8V DC), +12V DC, and +24V DC will be distributed to each module by regulating devices attached to the power bus.

3.2.5 Standardized Low-Cost/Low-Power Components

Building modular robots should be cheap in terms of dollars and power. Low-cost (less than $200), low-power complementary metal-oxide semiconductors (CMOS) devices (e.g., integrated circuits, regulators, microprocessors, etc.) will be used in the design of the standard hardware components. Off-the-shelf components should be used where possible. Battery-operated systems, in particular, must minimize power consumption wherever possible to maintain operation for extended periods. Standardization produces reusable, easily maintainable components that can typically be manufactured at a lower cost.

3.2.6 Unlimited Expansion

Above all else, the hardware architecture must be expandable. That is, provisions (hardware "scars") must be in place that will allow additional capabilities to be added at a later date. Application modules must be developed that can be assembled into a system solution with little or no integration overhead, that is, the time spent on integration is minimized. Expansion of cap ities is limited only by available technology and, as technology advances, so toc uld the abilities of a modular robot incorporating that new technology as an add-(module.

3.3 MRA SOFTWARE REQUIREMENTS

The general software requirement is for a computer architecture to support distributed processing among numerous functional modules that can effectively communicate with one another in a coordinated manner to accomplish a task in real time. The MRA provides software functions via standard libraries targeted at various processors that application programmers use to build these modules. Specific software requirements are listed below.

3.3.1 Distributed-System Software Services

Modular robots are distributed systems. The architecture must provide for distributed processing by means of software functions that, at a minimum, support interprocess communication and coordination. These functions will allow two or more modules to communicate with one another in an efficient manner, and will allow module processes to be coordinated such as tasks are coordinated in a multitasking environment. Response time, measured as the time from transmission of a typical length command packet to receipt of an acknowledgment of that packet, will be less than 100 μs.

3.3.2 Standard Module Application Controller

Each robot module is controlled by an application program. For modules that do not require a specialized controller (e.g., simple sensor or actuator systems), a standard (default) application controller will be provided. The controller will be responsible for managing the subsystems of the architecture to perform the function of the module. Operation of the controller is fixed and cannot be changed. Modules requiring more sophisticated control will replace the default controller with a specialized (user-supplied) application controller.

3.3.3 Standard Communications Protocol

Modules communicate with one another by using a protocol designed for high-level control of robot functions. A standard protocol will be used for interprocess communication, and for communication between external devices such as the control station. The protocol will support efficient transmission of commands and status information between modules. An existing protocol should be used if possible to promote interoperability between different systems. Also, adherence to the International Standards Organization (ISO) Open Systems Interconnection (OSI) model should be maintained with respect to a layered communications protocol (ISO, 1980).

3.3.4 Standard Device Interface

Devices are hardware components that interact with the real world to either measure (observe) it or effect (manipulate) it. Sensors and actuators, as well as the parallel

and serial ports of a computer are devices. Modules of a particular system are often constructed using similar components with similar interfaces. The architecture will provide a common interface to these and other devices. The standard device interface acts to decouple the details of the hardware implementation from higher-level software. Device "drivers" will be provided to support control of microcomputer/microcontroller devices such as serial ports, parallel ports, and timers. Drivers will be provided for a variety of target systems (e.g., 8031 microcontroller, STD-bus V20/8088/80286 microprocessors, etc.). A separate mechanism will also be provided for standard access to sensors and actuators, as well as to logical and virtual devices.

3.3.5 Flexible Control Methodology

The modular architecture does not specifically provide a formal method of control other than that offered by the standard module application controller. Programmers can develop their own control methodology to be implemented by using the subsystems of the architecture. The architecture must be flexible enough to support adaptation of new and existing control methodologies to modular robots.

3.3.6 Dynamic System Configuration (reconfiguration)

To support changing application requirements, modular robots must be able to be reconfigured at will with little or no software changes necessary. The architecture will provide a means for automatically recognizing the existence or absence of a module and to adapt system operation accordingly. The developer will be free to add or remove modules as desired without the need to change software module "addresses" or "identifiers". Configuration of the system with respect to software module interfaces will occur dynamically at initialization time.

3.3.7 Remote Function Operation

Each robot module can perform an associated set of functions. This applies to most systems developed under the modular architecture. Remote function execution is provided by the architecture so that one module can command another to perform some operation and return the results upon completion. The architecture is responsible for sending the appropriate command and for coordinating returned results with the original function call. Remote functions are used by application programmers in a manner similar to normal program function calls.

4.0 SYSTEM ARCHITECTURE

4.1 HARDWARE COMPONENTS

The MRA hardware architecture defines a MODBOT as a set of modules linked by a generalized distributed bus. The architecture does not require that the modules be arranged in any particular physical configuration, nor does the architecture specify the types of modules that a system must contain. Mobile systems, especially those involving human supervision, typically consist of a control station (CS), a remote platform (RP), and a telemetry link that connects the two. Note that the architecture does not force this decomposition upon an implementation, it is merely a convenient way of describing mobile systems involving human control.

4.1.1 Control Station (CS)

The CS is viewed conceptually as a MODBOT module whose central processing unit is located remotely from its associated module on the RP. The telemetry link connects the CS processor to the remote portion of the CS. Figure 10 shows a typical CS and its associated peripherals. The hardware architecture specifies only that the CS hardware be capable of supporting the MRA standard software systems.

4.1.1.1 Human Interface

The CS is the primary operator interface to the MODBOT. The CS acts as a control and display environment that allows the operator/developer to manipulate and observe any part of the robot and its world that is modeled by the system. The user can control operation of the robot at any one of several levels, e.g., teleoperated, reflexive, or supervisory depending upon the application software. Sensory and other state information is transmitted from the robot to the CS for display and analysis.

4.1.1.2 Displays

The CS uses one or more displays to present command, control, and sensory information to the operator. Realtime video transmitted from the RP can be displayed on separate monitors or integrated into a single-monitor system. Some applications may require more sophisticated devices such as headmounted displays that provide stereoscopic vision. The selection of displays is completely dependent upon the requirements of the application.

4.1.1.3 Controls

Possible controls include a speech-recognition device called the Voice Navigator by Articulate Systems, Inc. The Voice Navigator connects to the SCSI port of the Macintosh and employs a menu-driven voice training system to learn voice commands of a

specific operator. The audio speaker available on the Macintosh allows the CS to verify the command to tell the operator when the robot is done performing a task. The joystick selected for this application is the Advanced Gravis MouseStick. The Mouse-Stick features three programmable pushbuttons to allow for mouse-like menu control of specific functions, and is used to teleoperate (drive) the robot.

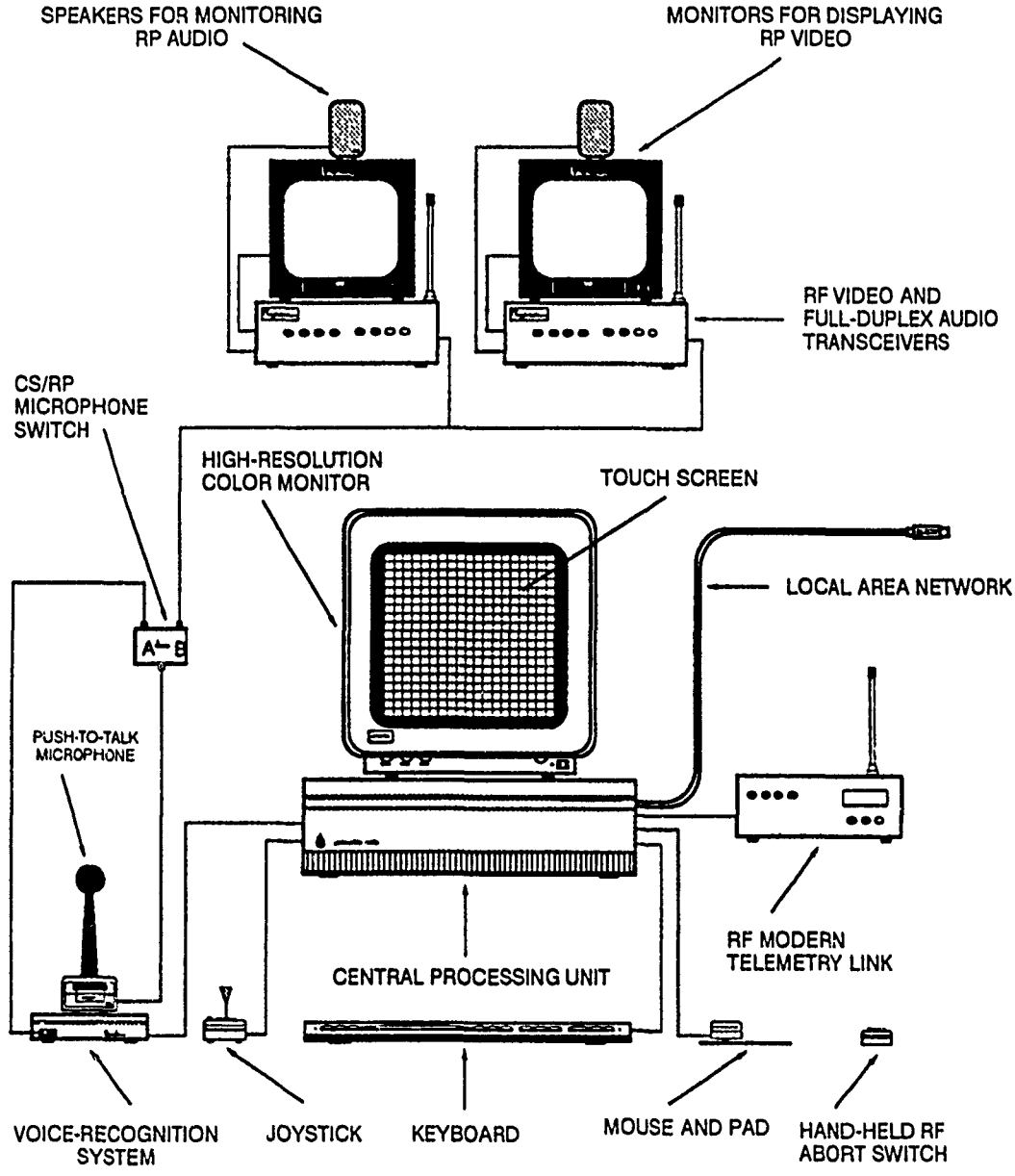


Figure 10. Control-station (CS) configuration showing external device connections.

4.1.2 Remote Platform (RP)

The RP, the remote portion of the MODBOT, represents what is normally referred to as the "robot". The RP is a series of modules that are connected together in a daisy-chain fashion by a distributed robot module bus. For mobile systems, the RP includes a propulsion system or base suitable for the application environment (i.e., land, air, sea, or other). The MRA specifies standard hardware interfaces between each RP module, and also indicates how the modules are attached to the base, electrically and mechanically. The architecture places no restrictions upon module arrangement.

4.1.2.1 Base

The base is comprised of the propulsion system and its accompanying power source. An outdoor robot would need a much more rugged platform and a fuel-driven system while its indoor counterpart would be most likely electrically driven. The only assumption the MRA places upon the base is that it must provide power to the rest of the system. For nonmobile robots, the base may consist simply of a mechanical module mount or frame and an electrical connection to a provided power source (e.g., a 24V DC transformer).

The base is viewed conceptually as an actuator with an associated mobility module that interfaces the base to the rest of the RP. The mobility module contains the standard MRA hardware components and is usually attached directly to the base.

For the in or security robot, a modified version of the TRC Labmate is being used. The wiring of the original platform was unacceptable and was totally redone. In addition, the original motor controllers were replaced with much more reliable and robust units. The Labmate, a stand-alone, 24V DC battery-operated base, has an RS-232 interface to an onboard processor that controls basic platform functions such as point-to-point transit and continuous-path control. High-level commands, such as "go forward 5 meters" or "turn 45 degrees," can be issued from a host processor via the RS-232 interface. The Labmate also has a joystick interface that is used to manually control the base.

4.1.2.2 Robot Module Bus (MODBUS)

Power, communications, and auxiliary control are distributed to each of the MODBOT modules via the robot module bus or MODBUS. The bus provides a standard interface to each module and simplifies the connections between modules. As shown in figure 11, the MODBUS consists of three sets of signal lines that form the standard module interface. The lines are broken apart at each module where the different sets of signals are run to the appropriate devices; the communications lines are connected to the intelligent-communications node, the power lines are attached to the power distribution node, and the auxiliary lines are attached to components as required by the particular module to which they run (the auxiliary control/feedback lines have yet to be

identified as to number and function, but will allow for a certain amount of growth in the MODBUS).

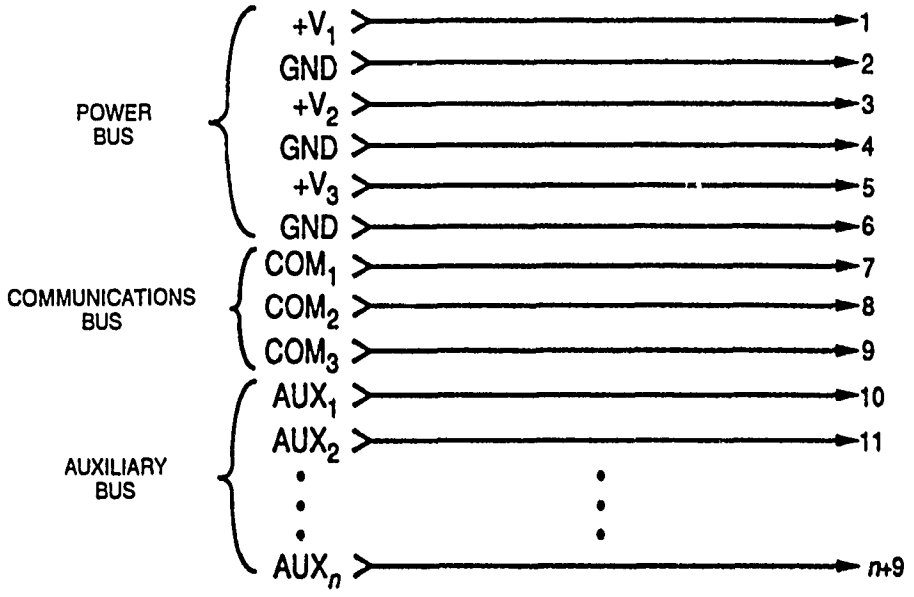


Figure 11. Generalized robot module "bus" (MODBUS).

The power lines carry common DC voltages (+24V, +12V, etc.) provided by the robot base and the power-conditioning unit. The communications lines support the intermodule local area network, while the auxiliary bus lines carry module-specific signals other than power and communications, such as video, audio, etc.

Modules are connected through the MODBUS, which is implemented on MOSER as a cable harness that is daisy-chained from one module to the next. This allows modules to be rearranged quickly and easily without any of the problems normally associated with relocating components of an embedded system (such as cable-handling nightmares).

4.1.2.3 Intelligent-Communications Node (ICN)

Each robot module contains an Intelligent-Communications Node (ICN) that provides a standard interface to the MODBOT local area network. The ICN manages medium-speed (>1 MBPS) data exchange between modules on the network by providing error detection/correction, collision detection, and general message passing services. Figure 12 is a block diagram of the ICN. The portion of the diagram contained in dashed lines may or may not be required. Single-chip microcontrollers are available that offer an integrated communications capability without the need for a separate network interface controller.

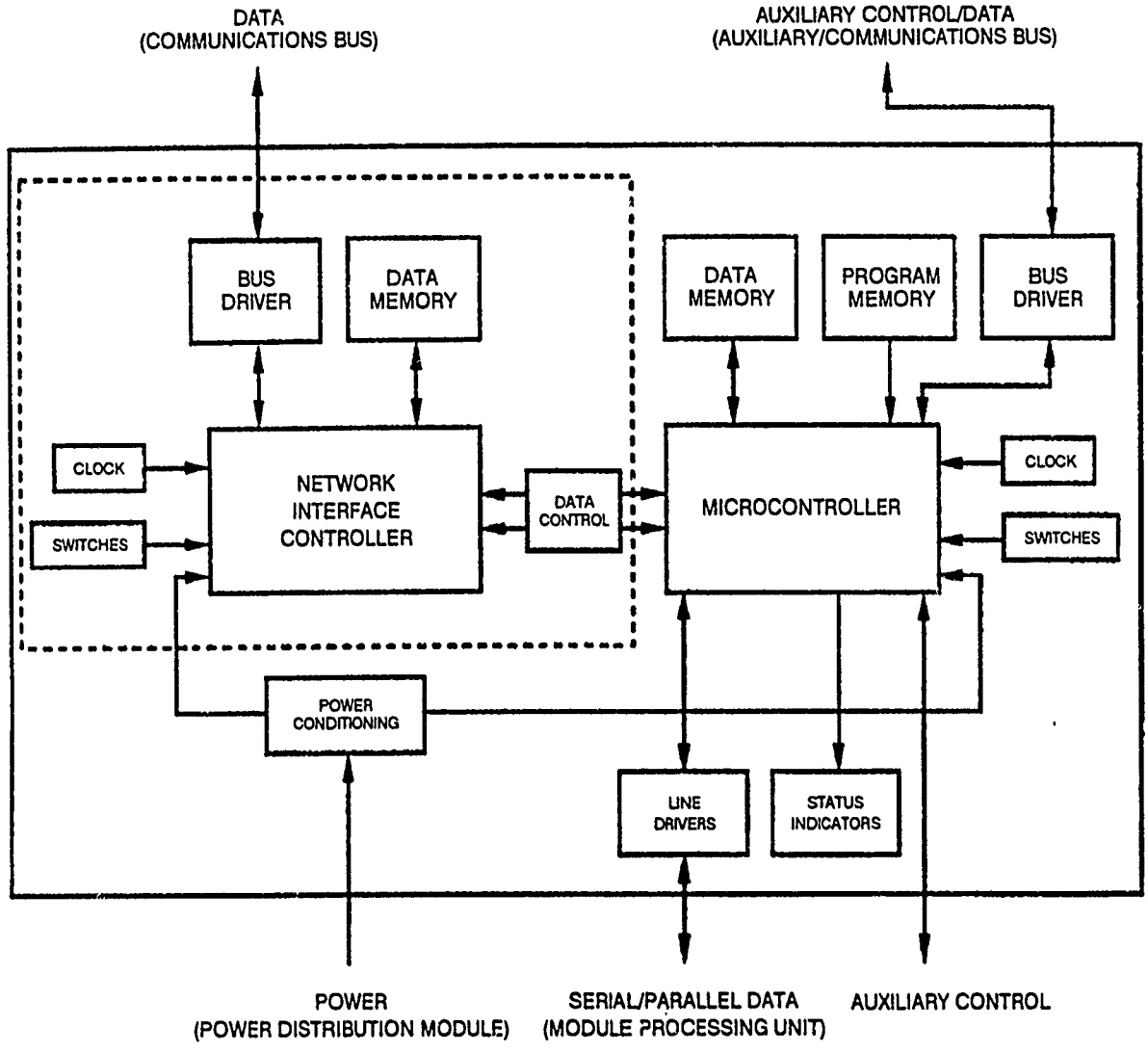


Figure 12. ICN block diagram.

The communications path is daisy-chained such that any module can talk to any other module—there is no central communications controller. Point-to-point as well as broadcast communications are supported. The ICN interfaces to the communications bus on one side and to the module processing unit on the other. The interface to the processing unit is configurable as either a standard RS-232 connection or as an 8-bit, high-speed, parallel connection.

The ICN on the MOSER is designed using CMOS components to minimize power consumption. An Intel 80C152 Universal Communications Controller is used as the ICN processor (Intel Corporation, 1989). The 80C152, an 8-bit microcontroller with an internal global-communications channel that implements the MODBOT local area network, can be configured to use one of four communications protocols: (1) Carrier-Sense Multiaccess with Collision Detection (CSMA/CD); (2) a subset of High-level

Data-Link-Control (HDLC); (3) Synchronous Data-Link Control (SDLC); and (4) a user-definable protocol. Baud rates of up to 1.8 MBPS are possible. A local-communications channel is also available for serial (RS-232) communications to the module's primary processor.

4.1.2.4 Power-Distribution Node (PDN)

Each robot module also contains a Power-Distribution Node (PDN) that provides local power conditioning, regulation, and short-circuit protection of power inputs from the MODBUS. Standard voltages such as +5V/+8V DC, +12V DC, and +24V DC are supplied by the PDN to the module. Distribution of power to individual components can be locally controlled via TTL-level inputs to "switches" on the PDN (figure 13). This allows specific subsystems to be turned off while not in use to conserve power. High-current thermal fuses are used in place of circuit breakers so that power that has

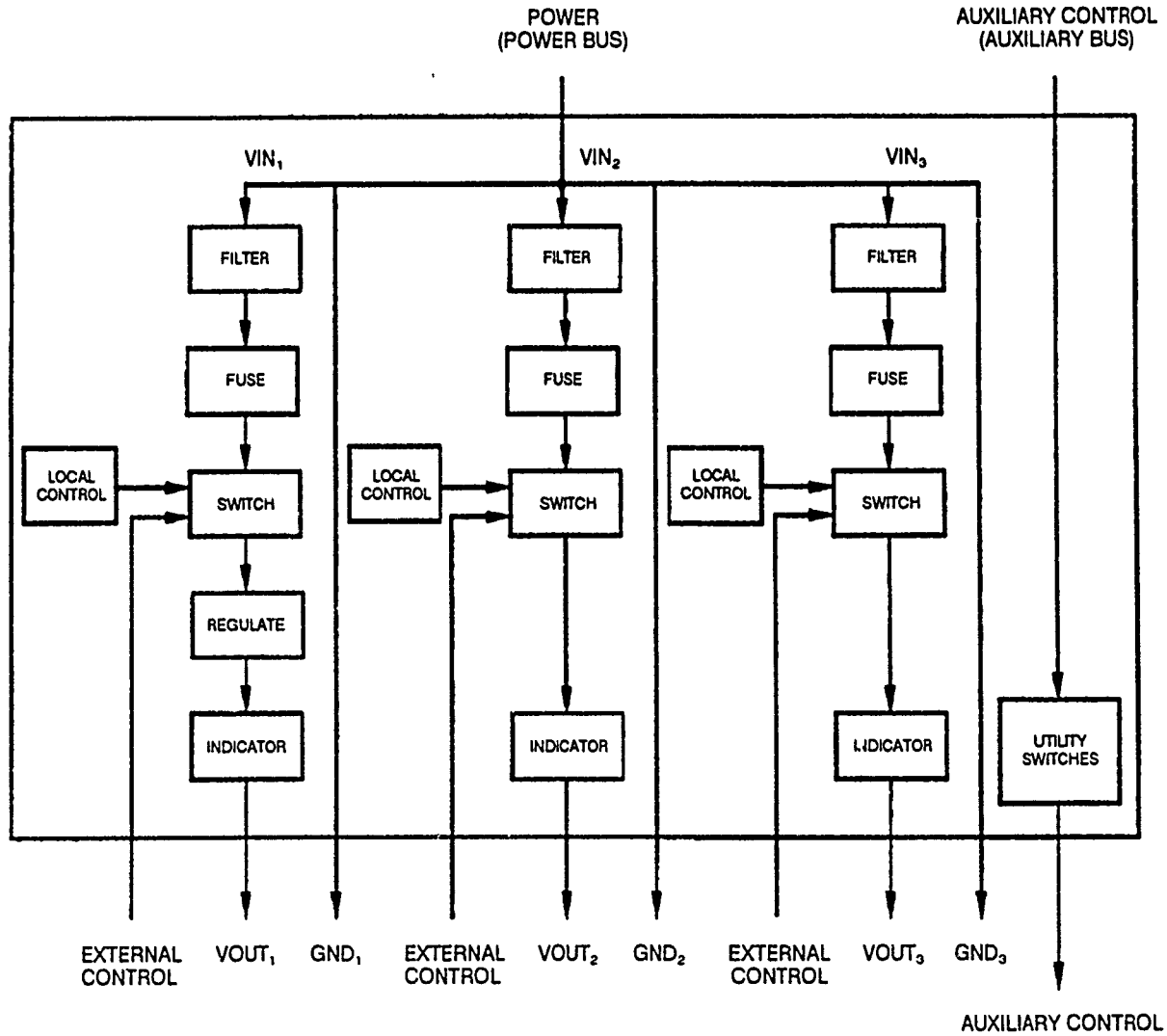


Figure 13. PDN block diagram.

been temporarily removed from a circuit will be automatically reapplied after a certain amount of time has passed. Thus, human intervention is not required to replace a blown fuse or reset a mechanical breaker, which is not always possible with a remote system.

Component selection (i.e., fuse size, regulator type, etc.) determines the current-carrying capabilities of each PDN and can be modified so that a particular module will be supplied with an appropriate amount of power. The MRA specifies limits on power consumption for each module so that a power budget for the entire system is achieved.

4.1.2.5 Platform-Power-Conditioning Unit (PPCU)

The Platform-Power-Conditioning Unit (PPCU) is responsible for converting the power supplied by the robot base to the standard voltages available on the MODBUS. The PPCU resides in the robot base and must be interfaced to the existing platform power system. It first protects and conditions the power supplied by the base with circuit breakers and spike suppressors, and then converts the power to standard voltages delivered over the MODBUS to the PDN. The supplied voltages are isolated from each other to avoid grounding problems. The only requirement that the PPCU places upon the platform is that it be capable of providing +24V DC at sufficient current. A block diagram for the PPCU is given in figure 14.

The PPCU also provides a standard interface between the robot platform and the power-distribution portion of the MODBUS. The PPCU makes the MODBUS base-independent, and decouples the power system of the base from the rest of the MODBOT.

4.1.2.6 Sensor, Actuator, and Processing Modules

A module is conceptually an independent unit with associated sensors, actuators, and/or processors. Each module must contain an ICN, a PDN, a central processing unit, also known as the Module Processing Unit (MPU) and, optionally, may contain sensors, actuators, additional processors or whatever else is required to support the function of the module (figure 15). The ICN is the module's interface to the robot local area network, while the PDN is the interface to the power system of the robot. The MPU implements the software systems of the MRA along with application-specific software provided by the developer or intelligent user. The only restrictions the MRA places upon the processing unit is that it be capable of implementing the MRA software subsystems, and capable of communicating with the ICN. Any processor meeting these requirements can be used, from simple single chip microcontrollers to sophisticated 68000-based microcomputers.

The ICN is connected to the MPU through either a standard RS-232 or an 8-bit parallel interface. Command and control data are transferred between the MODBUS local area network and the MPU via the ICN. The ICN receives power directly from the

PDN, and the ICN can control module power distribution through connections to local switches on the PDN.

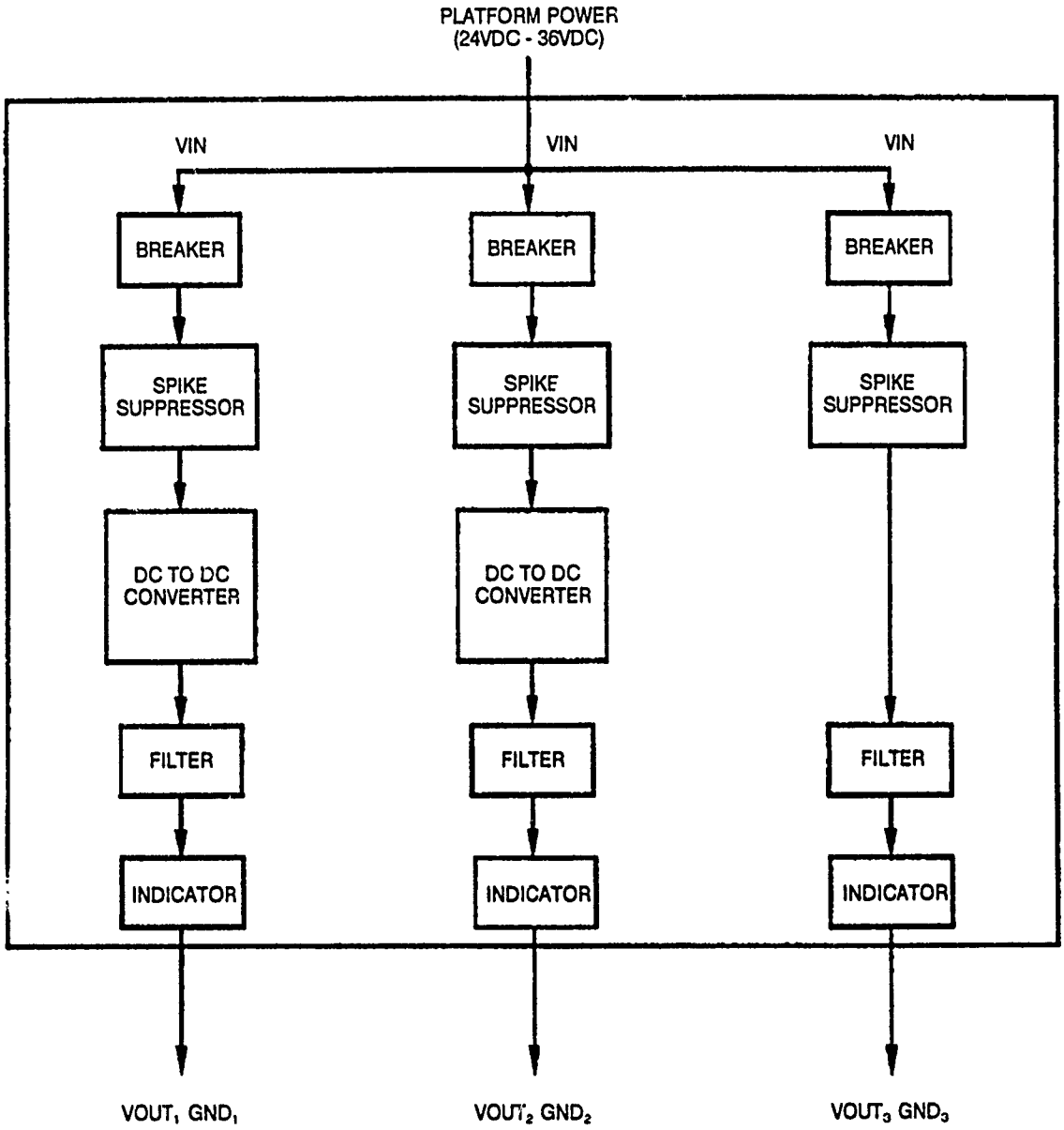


Figure 14. PPCU block diagram.

The MPU also receives power directly from the PDN, and can control module power distribution through connections to local switches on the PDN. The PDN is connected directly to the power portion of the MODBUS.

Modules interface to the real world through sensors and actuators connected to the module processor. A human interface—usually in the form of a serial connection—should be included on each module for diagnostic and test purposes. The environment interface shown in figure 15 refers to a possible connection to a secondary

communications system such as an I/R link or high-speed local area network stationed throughout the environment.

Construction and assembly of modules should be standardized so as to maximize component interchangeability and ease of connectivity. The MRA does not specify what modules are to be made of, it only specifies certain components that each module must include (the ICN and PDN for example). The physical implementation of a module will vary between applications; the module in figure 15 is a conceptual entity that can be implemented in a number of ways depending upon the user's needs.

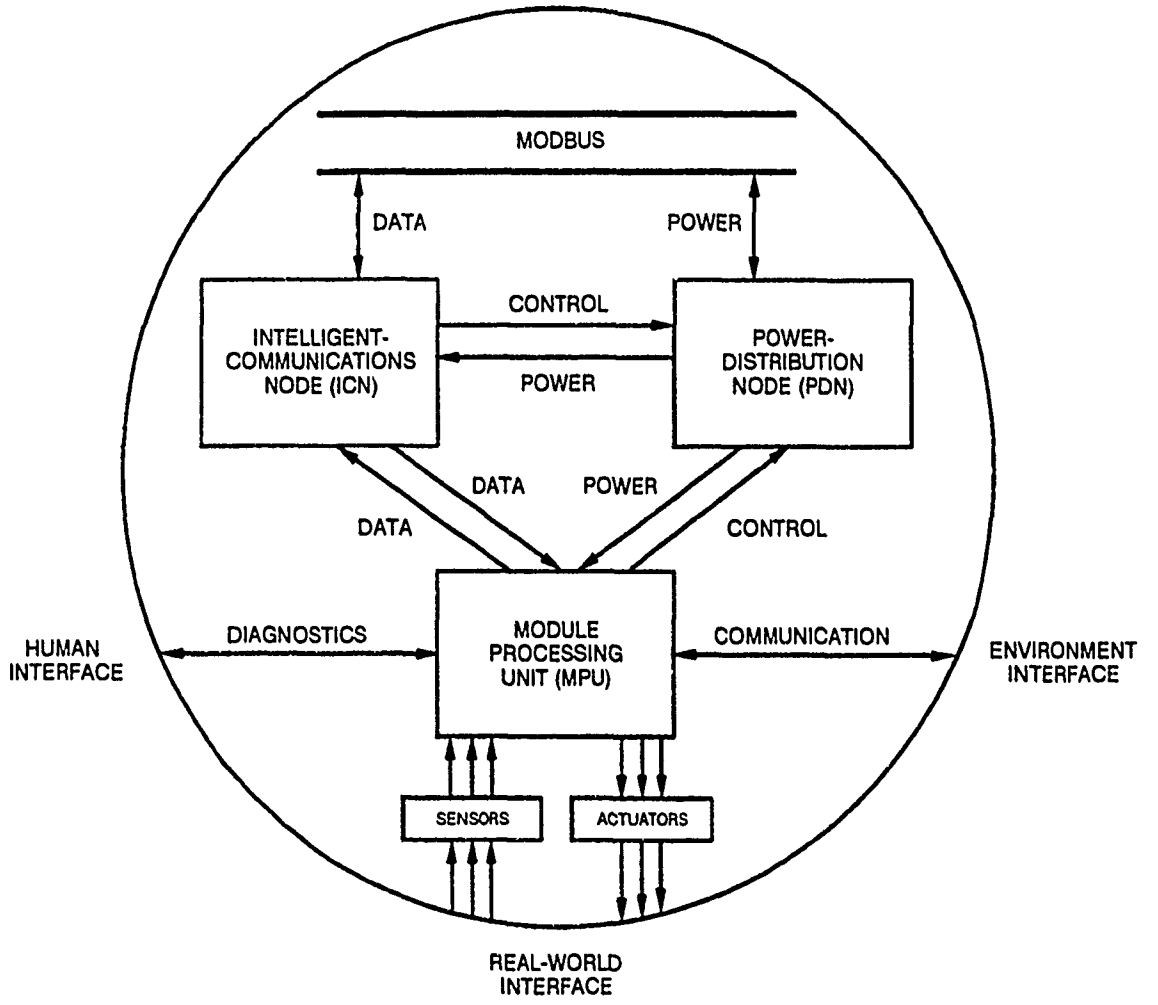


Figure 15. Generic robot module. The communications link to the external environment (environment interface) and the sensors and actuators (real-world interface) are optional.

Modules should have a single function or purpose. For example, a video transmitter should not be combined with an ultrasonic range sensor on a single module, rather the two should be implemented as separate units. This is equivalent to having distinct software modules that support separate functions of a program, and is consistent with modular-system-engineering practices.

Modules required for the mobile-security-robot application include actuator, sensor, and processing modules sufficient for intelligent navigation and for detection of "intruders." This includes a mobility module that interfaces to the robot platform, ultrasonic collision avoidance and navigation modules, an intrusion detection module, a near I/R proximity module used as a redundant-collision-avoidance sensor, an audio/ visual module used during teleoperation, a CS module that houses the telemetry link to the CS processing unit, and a high-level processing module that is responsible for path planning, world modeling, and overall system coordination.

4.1.3 Telemetry Link

The telemetry link is the external connection between the CS and the RP, and is application dependent. Typical command and control links (as opposed to video or audio links) use radio frequency (RF), infrared (I/R), or some form of tether such as a fiber-optic cable. A combination of devices can be used in situations where the capabilities of a single device is not sufficient for the given environment (an intelligent communications controller could then switch between systems as fields of coverage or signal strength change). The link allows the CS processor sufficient bandwidth to effectively transfer information to the robot (and vice versa).

The telemetry system, part of the CS module, is an external connection of the CS module-processing unit to the CS remote-platform ICN. Telemetry units are located both with the CS processor and within the CS remote-platform module, and allow information to be transferred indirectly between the CS processor and the RP local area network just as if the CS processor were located onboard the RP (figure 16).

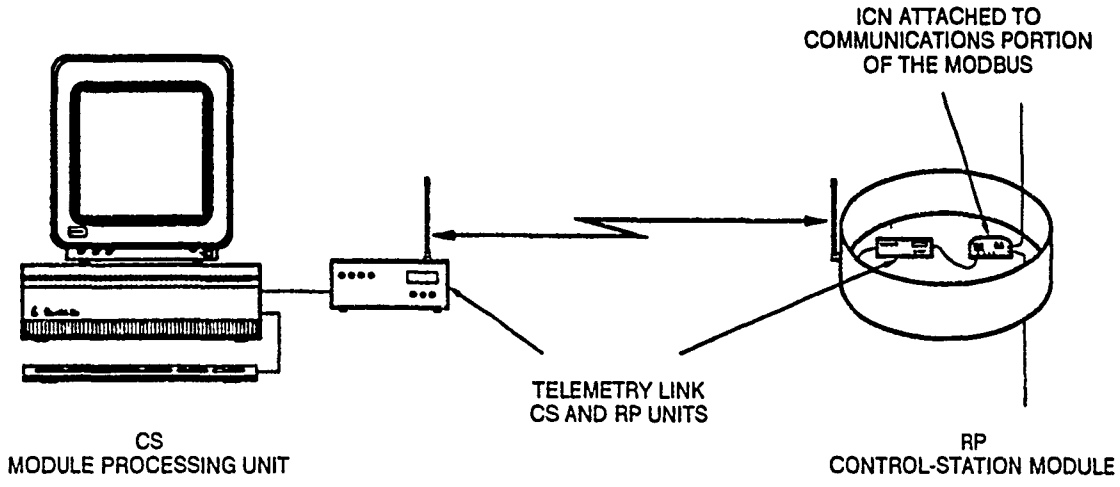


Figure 16. Telemetry link connecting the control-station (CS) processing unit and the remote-platform (RP) module.

MOSER uses a hard-wired tether that transmits command and control information between the CS processor and the remote CS module as an RS-232 signal operating at 9600 baud. In addition, the tether carries dual-channel audio and dual-channel video. Future plans call for replacement of the tether by an RF spread spectrum local area network controller for the command and control link and two RF video transmitters with dual-channel audio.

4.1.4 Communication Networks

Multipoint control—having one station control two or more RPs—is accomplished through the use of local-area wireless network (LAWN) modems. Data exchanged between the CS and the RP over the telemetry link contains addressing information that identifies the source and destination of transmissions. Each wireless modem has an associated address identifier. The modem controller examines incoming data and accepts them only if the address matches its own. By changing the destination address, a single CS can communicate with and control multiple MODBOTs. Figure 17 illustrates the concept.

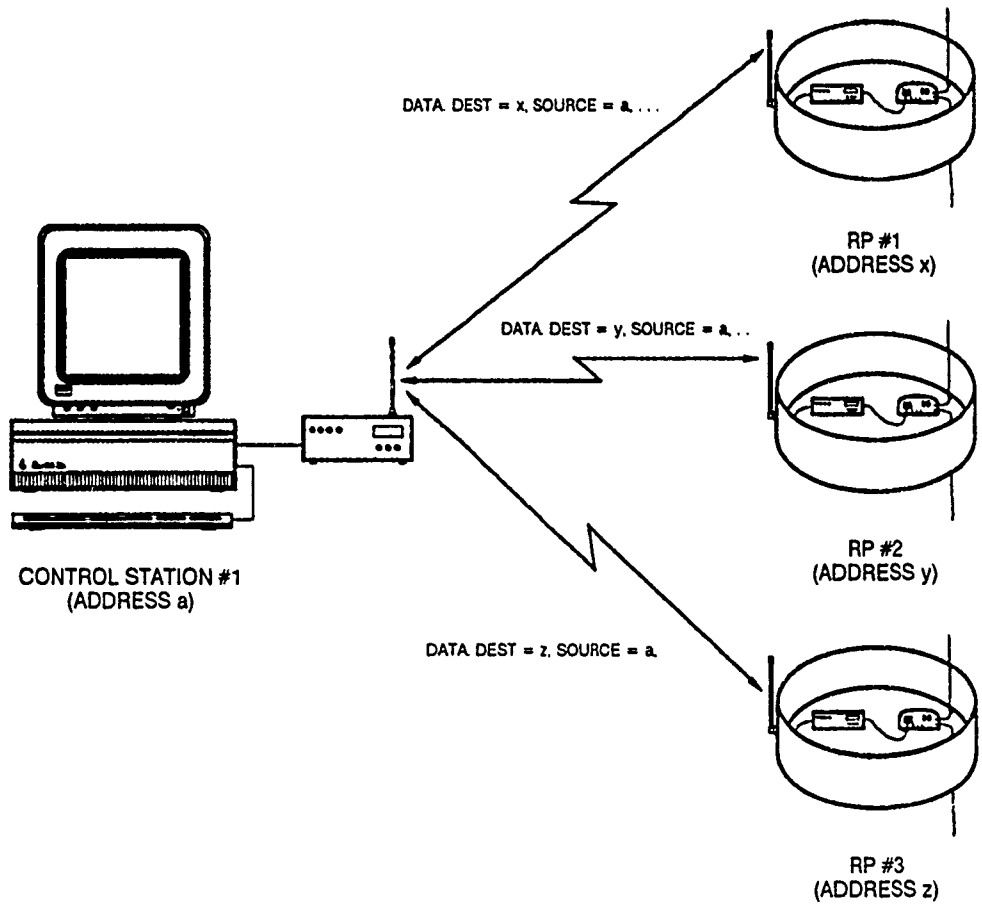


Figure 17. A single control-station (CS) controlling multiple remote platforms (RP) (note the addressing).

Multiple CSs can also be used to control a single RP. Each CS can be assigned a different subsystem to manage or monitor. Coordination between stations can be accomplished by communication over a separate local area network (figure 10).

4.2 SOFTWARE COMPONENTS

The MRA software architecture provides a framework around which distributed applications can be developed and implemented in a modular manner. The software systems are organized as layers in an *N,N-k* architecture where one software subsystem may have views (access) to multiple subsystems below it in the hierarchy (Lorin, 1988). Figure 18 is the modular-architecture-system image. The four primary software interfaces are given vertically on the right side of the diagram (e.g., Message-Manager Interface, LAN/MPU Interface, etc.) while the software subsystems that provide the interfaces are shown as layers within the hierarchy (e.g., METHOD LEVEL, COMMUNICATIONS LEVEL, etc.). MODBOT software application developers are able to use any subsystems as *desired*. However, applications must provide specific functionality that is established by the MRA and is otherwise obtained through the use of these subsystems. A standard software "library" is provided with functions available for inter-module communications, simple task control, and management of local and system module function execution. Software developers need only supply a minimal set of device-specific software "drivers" to implement the entire MRA software architecture (i.e., the software subsystems are portable between hardware implementations with only few modifications necessary).

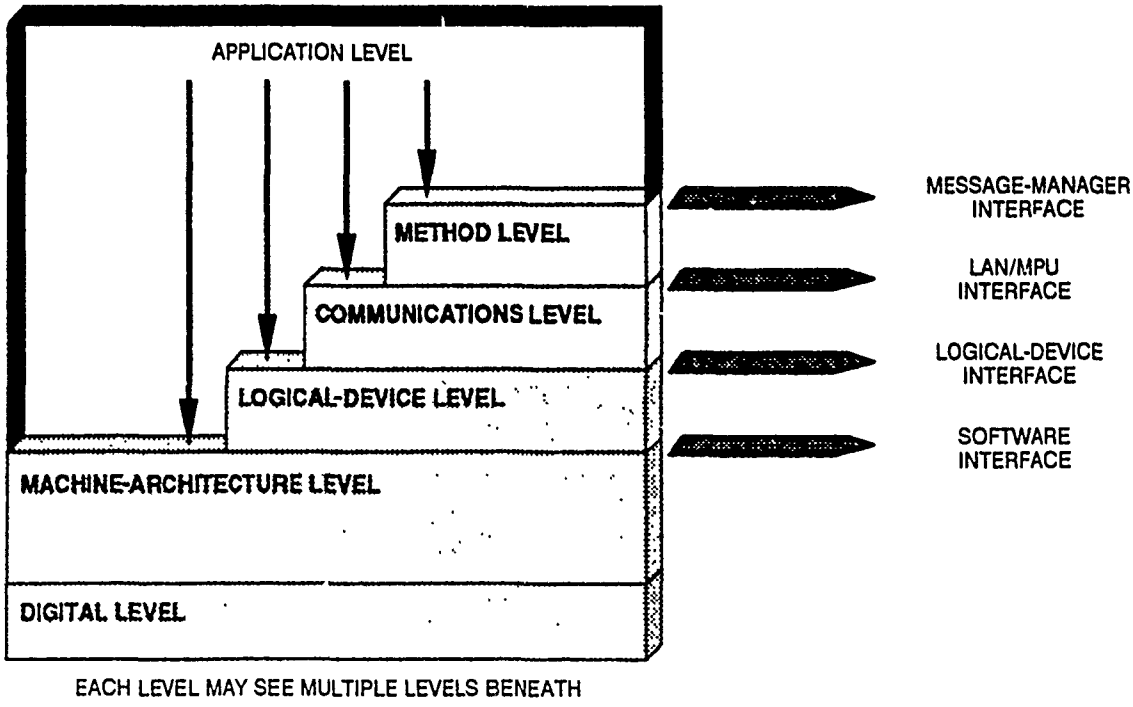


Figure 18. MRA system image (*N,N-k* architecture).

Each robot module can be thought of as an object that responds to a variety of commands represented by the object's methods and whose internal state is maintained by instance variables. Robot modules are of the class *Module* and possess certain capabilities common to all objects of that class. Modules also *inherit* the capabilities of their superclass (*Object*). Through classification and other properties of object-oriented programming, robot modules are given (by definition) common functionality.

The software subsystems of the MRA directly support object-oriented design and implementation of distributed, highly modular control systems for use on mobile (and nonmobile) robots. An object-oriented approach promotes both reusable software components and modularity at several levels. Additional capabilities can easily be added to a module simply by adding new methods to the object's class.

Below is a functional decomposition of the MRA software systems as implemented on the ICN and the MPU. (The ICN software is a subset of the MPU software with the exception of the Global Communications Subsystem, which is unique to the ICN.) Figure 19 is a block diagram of the MRA software subsystems as implemented on the MPU (note that the Global Communications Subsystem and the Logical-Device Interface are not shown).

4.2.1 ICN Software System

The ICN software system is responsible for managing medium-speed communication between the MODBOT local area network (LAN) and the MPU on board each module. The ICN provides a standard interface between the LAN and the MPU, and is the cornerstone of the MRA in that it provides for distributed MODBOT module communication and control.

The ICN software supports a 1.8-MBPS (maximum), CSMA/CD (peer-to-peer) communications network configured as a bus with deterministic access to a maximum of 255 modules (slots). There is no central or master-communications controller. Each node has equal access to the network and is responsible for managing its own resources. The Intel 80C152 global serial channel (GSC) is used as the node controller.

The ICN software system is composed of the following five functional units:

1) Global-Communications Device Handler.
2) Global-Communications Interface.
3) Local-Communications Device Handler.
4) Local-Communications Interface.
5) Intelligent-Communications Controller.

The Global-Communications Subsystem and the Intelligent-Communications Controller are unique to the ICN software system. The Local-Communications Subsystem is common to both the ICN and MPU processing systems.

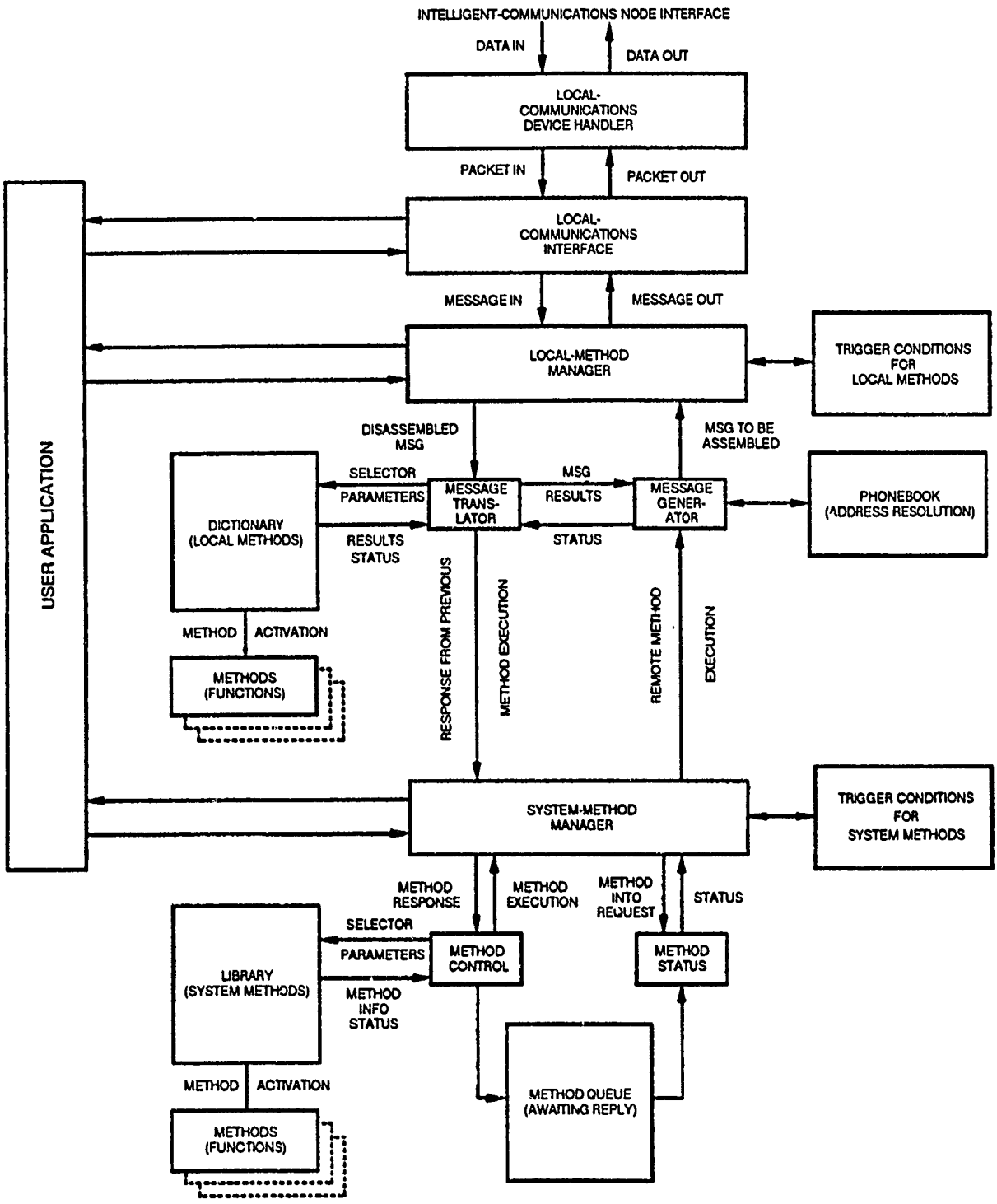


Figure 19. MRA-software block diagram. (Global-Communications Subsystem and Logical-Device Interface not shown.)

4.2.1.1 Global-Communications Subsystem

The Global-Communications Subsystem implements a standard set of functions on the target LAN hardware. These functions are used by higher-level software to access

the LAN. The subsystem is broken down into the Global-Communications Device Handler (responsible for low-level control of the LAN hardware) and the Global-Communications Interface (which implements the standard network access functions available to other software subsystems).

The Global-Communications Device Handler interacts directly with the ICN hardware to provide external communications between the LAN and the ICN. The functions at this level are device-specific, and must be modified for the target (LAN) hardware. The functions provided by the GSC and the device handler implement the Physical Link Layer and the Data Link Layer of the ISO open-systems communication model (International Standards Organization, 1980).

The device handler logically decouples the standard functions of the Global-Communications Interface from the hardware implementation. To use a device other than the GSC as the MODBOT LAN, only the Global-Communications Device Handler functions must be rewritten.

The Global-Communications Interface is an abstraction of the lower-level functions and represents the user interface to the LAN. Functions at this level are completely hardware independent, and provide services for initializing and transmitting messages to/from the LAN.

4.2.1.2 Local-Communications Subsystem

The Local-Communications Subsystem implements a standard set of functions on the target-processor communications hardware. These functions are duplicated on both the ICN and MPU, and are used by higher-level software to communicate between the two systems. The subsystem is broken down into the Local-Communications Device Handler (responsible for low-level control of the serial or parallel communications hardware) and the Local-Communications Interface (which implements the standard communications port access functions available to other software subsystems).

The Local-Communications Device Handler is responsible for low-level data exchange between the MPU and the ICN. Functions at this level deal directly with the target hardware and are device-specific, so their implementation will change as the hardware changes. The device-handler functions are distributed between both the MPU and ICN, providing the communications link between the two systems.

The device handler logically decouples the higher-level functions provided by the Local-Communications Interface from the specific hardware implementation. Changing serial-communications controllers, for example, requires only that certain portions of the physical interface be modified to maintain consistency and compatibility at higher levels.

The Local-Communications Interface is an abstraction of the lower-level functions and represents the user interface to the interprocessor (local) communications port.

Functions at this level are completely hardware independent, and provide services for initializing and transmitting packets to/from the serial or parallel port.

4.2.1.3 Intelligent-Communications Controller

The "main" program of the ICN is the Intelligent-Communications Controller whose MPU counterpart is the User Application. The ICN controller is very simple: it initializes the Local- and Global-Communications Subsystems, and then coordinates transmission of information (data packets that represent module messages) between the ICN and the MPU. Messages are received from the network and passed along to the MPU. Messages are also received from the MPU and then sent on to the network for distribution as appropriate.

Currently, the communications controller is a simple message buffer between the MPU and the MODBOT LAN. Future plans include adding the message recognition and response capabilities of the Message Manager to the ICN for more sophisticated control. This would make the ICN software nearly identical to that of the MPU.

4.2.2 MPU Software System

The MPU software system is responsible for execution of both local and system methods (functions) as directed by the application module. The MPU provides the User Application with standard interfaces to the message-passing, function-execution, and logical-device-control facilities of the MRA.

The MPU software system is divided into two distinct components: the MRA MPU standard software services, and the MPU application program, which is the main routine supplied by the developer. A default controller is supplied by the MRA (for simple applications) that replaces the main program.

The MPU software system is composed of the following six functional units:

1) Local-Communications Device Handler.
2) Local-Communications Interface.
3) Local-Method Manager.
4) System-Method Manager.
5) Logical-Device Interface.
6) Application Controller.

The Message Manager, Logical-Device Subsystem, and the Application Controller are unique to the MPU. The Local-Communications Subsystem is common to both the MPU and ICN, and was described previously in section 4.2.1.2.

4.2.2.1 Message Manager

The Message Manager is responsible for translating and executing incoming messages, and for generating messages in response to external and internal requests. The

Message Manager is also responsible for external message address resolution, which relies on the ability to automatically determine the status of a module on the MODBOT network.

Functions that describe the behavior of a module are called *local methods* and are referred to as "internal." The functions available to all modules are called *system methods* and are referred to as "external." A dictionary contains compiled versions of the local methods that are executed upon receipt of the appropriate message. A *library* holds references to all of the system methods that an MPU needs for its application.

The Local-Method Manager coordinates receipt of incoming messages and their interpretation. Messages that request action or information are activated as local methods contained in the dictionary, while messages that are responses from previous external requests are passed on to the System-Method Manager.

The System-Method Manager coordinates activation of and response to system methods. A *method queue* is maintained by the System-Method Manager for external commands or data requests that require a response. Upon receipt of the required information, the method queue is searched according to message address and sequence number, and the external reference is resolved with the response being passed back to the calling function. The queue allows the application program to activate several methods sequentially and then coordinates receipt of responses, allowing the main program to continue execution until it is ready to process the incoming data. Services are available to the application program for examining the status of queued-method activations.

Initialization of the Message Manager includes resolving system-method address references. A *phone book* is maintained that contains the names of all externally referenced modules. Upon startup, each module whose name appears in the phone book is searched for on the MODBOT network. If found, the module's address is entered and subsequent references to that module can be resolved. If the module can't be located, then system methods referencing that module will fail.

A *trigger-condition table*, for both local and system methods, allows for execution of functions according to qualifiers placed on local (instance) variables. Functions can also be activated by an expiring timer with a given period (typically specified in milliseconds). Conditions for system methods are preprogrammed by the applications developer, while local-method conditions are set by external commands.

4.2.2.2 Logical-Device Subsystem

The logical-device subsystem consists of a logical-device interface and an associated *blackboard* data structure. The blackboard is used as a global module data storage and retrieval mechanism, and provides a convenient and consistent means of maintaining local variables (Aviles, Laird, & Myers, 1988).

The blackboard is based upon *logical devices* that have an abstracted real-world implementation. Logical sensors and actuators, for example, are used to represent devices whose state is maintained in the blackboard. Functions attached to the logical devices update their physical counterpart as information is requested from or entered into the blackboard. Devices that have no hard implementation are called *virtual*, and can be used to simulate an actual entity.

The logical-device interface provides software services for adding items to the blackboard, and for updating and retrieving the various data fields of those items. Activation of the functions attached to the items is automatic depending upon the device interface function used.

4.2.2.3 Application Controller

For applications that require no special processing (e.g., simple sensor or actuator modules), a default application controller is provided by the MRA MPU. The default controller takes the place of the User Application main program, and is responsible for initializing and coordinating the other subsystems of the MPU.

For specialized modules such as high-level path planning or distributed task controllers, the user must supply the main application program (i.e., the User Application). In this case, the application program is responsible for initializing the MPU subsystems and for managing the MPU software resources as required (see section 2.2.3 for a description of other applications).

4.2.3 Intermodule Message Format

The MODBOT network is based upon the ISO open-systems communication model, and implements the physical, the data-link, the presentation, and the application layers. The message format used at the presentation and application levels is based upon the Robotic-Vehicle Message Format (RVMF) developed by TACOM (Brendle, 1990). The primary differences between the RVMF and that used under the MRA is in the placement and bit requirements for the unit destination and source address fields as well as the message length and sequence number. These fields were modified to optimize message acknowledgment and function execution (only three bytes are required to ACK a message and only five bytes needed to execute a module function). The RVMF block address and unit ID correspond to the MODBOT address and module unit ID respectively.

The modified RVMF message format (RVMF*) is (nearly) maintained from layer to layer, that is, very few overhead bytes are added as the message is passed between the data link and application layers. This greatly increases data throughput and simplifies the MRA communications software interfaces. Figure 20 is a preliminary definition of the modified RVMF communications protocol. The figure does not break the protocol

down into the multiple OSI layers. A message checksum or cyclic-redundancy check (CRC, not shown) would be added by the data-link layer before transmission.

DEST MODBOT ID	DEST UNIT ID	SOURCE MODBOT ID	SOURCE UNIT ID	SEQUENCE NUMBER	TRANSACTION DISPOSITION	TRANSACTION CATEGORY	FUNCTION ID	PARAMETER LENGTH	PARAMETER
DESTINATION ADDRESS		SOURCE ADDRESS		MESSAGE COORDINATION		FUNCTION SELECTION		FUNCTION PARAMETERS/RESULTS	

Figure 20. MODBOT communications protocol (multiple layers).

4.2.4 High-Level Module Definition

The MRA provides facilities for describing modules at a high level of abstraction. The module description is then translated into compilable code that can be included in the application. A syntax similar to the C programming language is used to define a module's methods as well as internal instance variables. Classes as well as objects can be defined. Figure 21 is an example of an object definition for an I/R proximity (I/RP) module.

```
with OBJECT;
with MODULE;
module IRP
{
        var OBJECT:
                char        Class[]                 = "MODULE";
                char        Superclass[]            = "OBJECT";

        var MODULE:
                byte        Address                 = 0;
                char        Name[]                  = "IRP";
                bit         Subsystem_Power         = OFF;
                int         Operational_Status      = IDLE;

        var IRP:
                int         IRP_Max_Update_Rate     = 10;       /* Hz      */
                int         IRP_Number_Sensors      = 11;
                int         IRP_Sensor_Range        = 30;       /* Inches  */

        method QUERY_OPERATIONAL_LIMITS:
                int         IRP_Max_Update_Rate();
                int         IRP_Number_Sensors();
                int         IRP_Sensor_Range();

        method STATUS_REQUEST, PERIODIC_STATUS_REQUEST:
                bit         IRP_Sensor_Report(int S : in);
                bit*        IRP_n_Sensor_Report(int S1, int S2 : in);
}
```

Figure 21. Example of an MRA definition for an I/R proximity module.

5.0 SYSTEM OPERATION

5.1 GENERAL PHILOSOPHY

Modular-robot operation depends upon the application and is normally the responsibility of the system developer. However, if the standard module application controllers are used, then operation of the robot is based upon the combined behavior of each module as implemented by the individual module functions. That is, the default controllers simply respond to incoming commands by executing the functions provided by the application programmer (section 4.2). If the standard controllers are not used, then operation is determined by the control methodology implemented by the developer. In this case, the developer is free to operate the robot however desired, and simply uses the hardware and software components of the MRA to implement the design.

5.2 AUTOMATIC CAPABILITIES

Independent of the approach above, all modular robots will have certain inherent capabilities that will automatically execute at system initialization. This includes built-in tests (BITs) for diagnostic purposes, and module identification and address-resolution functions for self-configuration. These capabilities are implemented at the lower levels of the hardware and software architecture and cannot normally be overridden or defeated.

5.2.1 Self-Diagnostics

Upon power up, the standard software systems (for example, the global- and local-communications systems) perform a variety of diagnostic tests to ensure the integrity of the hardware and software components. Errors are reported to the higher level subsystems for action. In case of a severe error, degraded performance is preferable over total loss of capability and will be attempted if possible. Certain failures will prevent a module from operating altogether, such as a low-level-communications-driver failure. Diagnostic indicators will be present on all standard hardware components such as the ICN, PDN, and PPCU.

5.2.2 Self-Configuration

Each robot module has an associated network address that is set by hardware and is read by initialization software. References to modules must be correlated with their addresses as in resolution of external function references in the compilation of computer programs. Address resolution takes place automatically upon power up by high-level software subsystems of the MRA. Unresolved references occur when a module

cannot be located on the network, in which case an error is flagged. The process is dynamic and allows the robot to configure itself each time power is applied (or the system is reset).

5.3 SYSTEM COMMUNICATION

There are four levels of communication associated with modular robot control: (1) communication between the ICN and the robot LAN (on the MODBUS), (2) communication between the ICN and the MPU, (3) communication between the CS and the RP, and (4) communication between multiple CSs. These levels are shown in figure 22. (This topology does not apply to robots that do not have an associated CS.) When more than one CS or modular robot is employed at the same time, configurations can be conceptualized that take advantage of the multiple communication pathways. (The communications protocol addresses MODBOTs as well as individual modules, that is, both the MODBOT and the module are identified in the address.)

MODBOT Teams (LAN communication)

A MODBOT team consists of one or more CSs controlling two or more modular robots. The CSs are connected using a local area network (LAN) located throughout the workspace (figures 22 and 23). The application programmer is responsible for coordinating control between the multiple stations (not a trivial problem). MODBOT teams address problems such as physical security of very large spaces such as a warehouse.

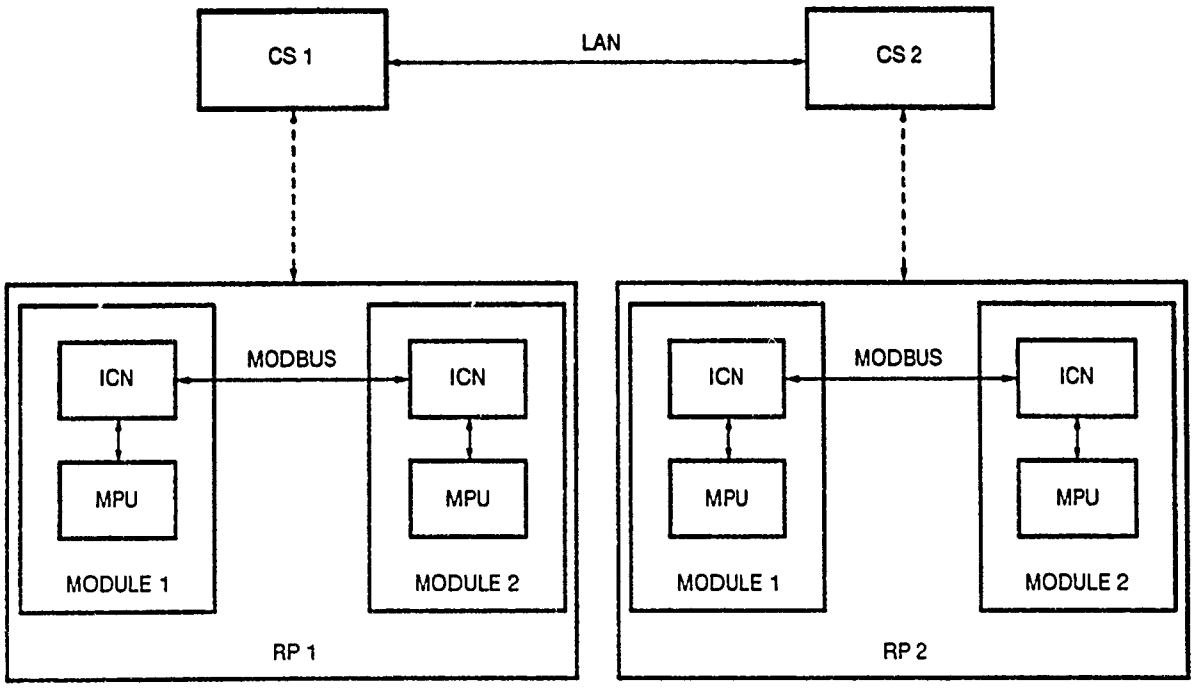


Figure 22. Communication paths between modular robot components.

MODBOT Divisions (WAN communication)

A MODBOT division consists of multiple MODBOT teams (figure 23). A wide area network (WAN) connects teams located at remote sites. Coordination at this level is very complex and may require separate CS dedicated for this purpose. MODBOT divisions can be used for applications such as inventory monitoring and control at several (possibly distant) sites.

5.4 OPERATION AS A SECURITY ROBOT

MOSER, the mobile-security modular robot, is responsible for autonomous navigation of an enclosed area such as an office building, and for the detection of intruders within that space. MOSER is modally operated, that is, one of several control modes is entered by the operator at the CS, and the robot behaves according to rules governing the selected mode.

Four logical levels of control are implemented on MOSER: teleoperated, reflexive, supervisory, and autonomous. Teleoperated control allows the user to directly navigate the MODBOT throughout its surroundings while monitoring sensory feedback on the CS displays. Teleoperation gives the operator full control of all MODBOT actions, including running the MODBOT into a wall as an extreme example. Reflexive control is a variation of teleoperation in which the MODBOT is now responsible for maintaining primitive reflexes that are conditioned by the operator. This prevents the MODBOT from running into walls. Supervisory control is a form of semiautonomous behavior. At this level, the operator is able to issue simple commands to the MODBOT and is able to instruct the MODBOT to traverse a path. Under supervisory control, the operator can intercede at any point and override MODBOT automatic functions. Under autonomous control, the operator need only specify goals to be reached or simple tasks to be executed. Autonomous control is attained with MOSER by its ability to operate as a self-sustaining mobile security robot that operates with a predetermined set of goals and responsibilities (e.g., identify and alert the operator to the presence of intruders).

A typical scenario would involve an operator manually piloting the robot to a starting point such as a door into a main hall, and then instructing the robot to patrol a path around the perimeter of the building and to sound an alarm if an intruder is detected within the area covered by specific sensors.

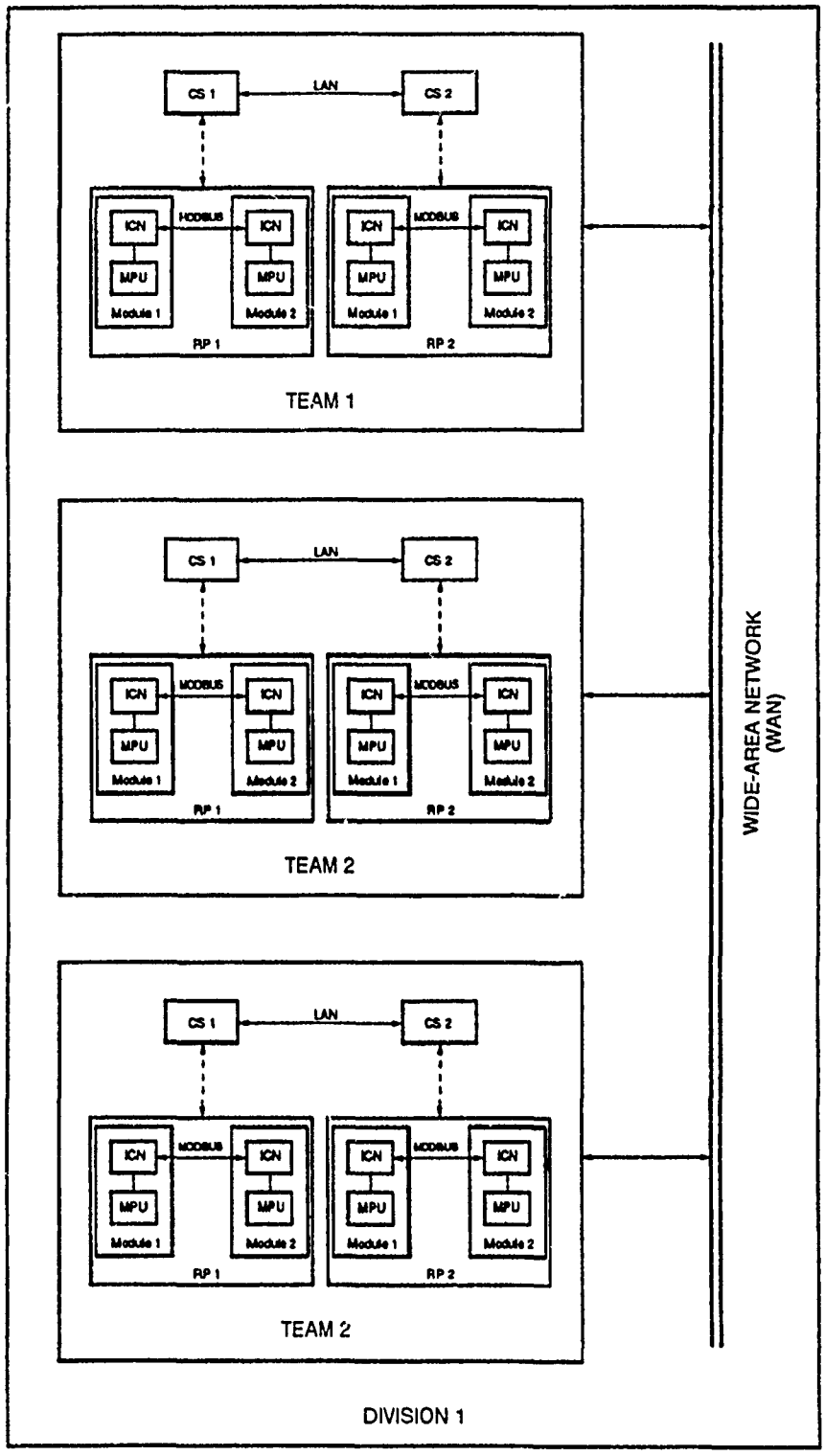


Figure 23. MODBOT teams (one or more MODBOTs) and divisions (one or more teams).

6.0 SYSTEM DEVELOPMENT

6.1 DOCUMENTATION

Development of the MRA follows standard software engineering practices as best as possible given a finite amount of time, money, and patience. The general plan is to research, design, review, and implement a modular architecture that meets an ever-changing array of requirements. Because the MRA is intended to be a standardized architecture—a common interface and control system that is shared among several agencies—emphasis is placed upon using "standard" (widely available or easily attainable) equipment, tools, and procedures so as to facilitate implementation of the "standard."

System documentation is a major portion of this effort and describes all aspects of the modular architecture and its development, from conception to implementation. This document is a high-level conceptual introduction to the MRA; it describes the preliminary architecture design and outlines an example application (i.e., a mobile security robot).

6.2 DEVELOPMENT EQUIPMENT AND STANDARDS

In developing the hardware and software systems of the MRA, considerable thought was given to the selection of development and target system equipment—making use of existing equipment was of major importance. For example, the decision to use a particular microcontroller as a (standard) module processing unit was initially based upon the availability of an existing cross compiler. In addition, the development process (especially implementation) can be simplified by trying to standardize on certain components and processes that are repeated throughout the system. Using a standardized computer programming language, for example, increases software portability and reusability.

The sections below list the software and hardware development tools and equipment that are being used in this project. The list is provided as background information and as a convenience for future MODBOT developers.

6.2.1 Software Development (Computers, Languages, etc.)

Below is a list of the major software tools and equipment used in developing the MRA. Only items that were used as "standard" equipment are listed.

Development systems

IBM PC-AT:

Software development.
Project documentation.
MS-DOS 3.2 and greater.

Macintosh IIx:

Software development.
Project documentation.
MultiFinder 6.03.
GRAVIS MouseStick GMPU joystick.
Farallon MacRecorder audio digitizer.
Articulate Systems Voice Navigator voice recognition system.

Programming language

Microsoft C compiler V5.1 (C programming language):

Path Planning module software development (IBM).

Franklin C-51 compiler (C programming language):

MPU software development (IBM).
ICN software development (IBM).
Application module software development (IBM).

Symantic Think C compiler (C programming language):

CS software development (Macintosh).

Software development tools

Documentation:

Wordstar 4.0 (IBM).
Microsoft Word 3.02 (Macintosh).
Cricket Draw 1.1.1 (Macintosh).
MacDraw II (Macintosh).

Other

Laplink(Mac) 2.0 (file transfer and data conversion).
Apple File Exchange .

6.2.2 Hardware Development (Computers, Platforms, etc.)

Below is a list of the major hardware components used in developing the MRA (and MOSER). Only components that were used as "standard" equipment are listed.

Target computer systems

CS:

Macintosh IIx.

MPU:

80C31 microcontroller.
Winsystems STD bus SBC8-8 (V20 processor).

ICN:

80C152 microcontroller.

High-level processing module:

Winsystems STD bus STD-AT (80286 processor).

Platform

MOSER:

TRC Labmate (modified extensively inhouse).

Module development

Robot module "ring" material:

1/8"–1/4" thick, 18" round plexiglass.
1/2" thick, 18" diameter PVC pipe.

6.3 INDEPENDENT DEVELOPMENT OF MODULES

An important aspect of the MRA is the ability to develop robot modules independent of normal system development and integration. The intention is that cooperating agencies independently develop MODBOT capabilities that can be easily integrated into an existing system or that can be coupled to form pieces of a new system. Modules are developed according to MRA standard interface requirements with the developer supplying the necessary hardware and software device-specific drivers. The concept is similar to third-party development of expansion or peripheral boards for the personal computer with the potential for product diversification as seen in this market applicable to the development of MODBOT modules.

6.3.1 Types of Modules That Should Be Developed

There are three major categories of MODBOT modules to be developed: (1) actuator modules, (2) sensor modules, and (3) processor modules. Development of specific modules is, of course, application dependent. Below is a brief listing of potential modules appropriate for indoor security applications.

Actuators

Actuator modules provide interaction with the physical environment:

1) Pan/tilt module for directional sensors such as cameras.
2) Manipulator module for grasping and activating controls.
3) Deterrence module for halting intruders.

Sensors

Sensor modules provide information for world modeling:

1) Environmental module for temperature, humidity, etc.
2) Intrusion-detection module having several different sensors.
3) Ultrasonic-ranging module for mapping the environment.
4) I/R-collision-avoidance modules for navigation.
5) Ultrasonic-collision-avoidance module also for navigation.
6) Video-motion-detection or vision-sensor module.

Processors

Processing modules provide system coordination and planning:

1) Vision-processing modules for navigation/object recognition.
2) Neural-network processing module for data reduction/analysis.
3) IBM AT processing module for path planning.
4) CP-31 microcontroller for use as a platform-interface module.
5) Control-station module for use as a human interface.

6.3.2 Adherence to Standard Interface Specifications

In developing MODBOT modules, adherence to the interface requirements specifications as given in the IRS is mandatory. Successful system integration is wholly dependent upon strict conformance to the MRA standards especially when multiple activities are involved. When possible, fabrication and distribution of standard architecture components should be done by the coordinating agency to ensure compatibility.

7.0 DEFINITIONS, ACRONYMS, AND ABBREVIATIONS

AMR	Autonomous Mobile Robot
AMRF	Automated Manufacturing Research Facility
ARDEC	Armament Research, Development, and Engineering Center
BIT	Built-in Test
CMOS	Complementary Metal-Oxide Semiconductor
CS	Control Station
CRC	Cyclic-Redundancy Check
CSMA/CD	Carrier-Sense Multiaccess with Collision Detection
GATERS	Ground-Air Telerobotic Systems
GRPA	Generic Robotic Processing Architecture
GSC	Global Serial Channel (of the Intel 80C152)
HDLC	High-level Data-Link Control
HDS	Hardware-Design Specification
ICN	Intelligent-Communication Node
IED	Independent Exploratory Development
I/R	Infrared
IRS	Interface-Requirements Specification
ISO	International Standards Organization
LAN	Local Area Network
LAWN	Local-Area Wireless Network
MB	Megabytes
Mbps	Megabits-per-second
MODBOT	Modular Robot
MODBUS	Robot Module Bus
MOSER	Mobile Security Robot (first application of the MRA)
MPU	Module Processing Unit
MRA	Modular Robotic Architecture
NASREM	NASA/NBS Standard Reference Model
NBS	National Bureau of Standards (a.k.a. NIST)
NHC	Nested-Hierarchical Controller
NIST	National Institute of Standards and Technology
NOSC	Naval Ocean Systems Center
OSI	Open-Systems Interconnection
PDN	Power-Distribution Node
PPCU	Platform-Power-Conditioning Unit

RCS	Realtime Control System
RP	Remote Platform
RVMF	Robotic-Vehicle Message Format
SBIR	Small-Business Innovative Research
SDLC	Synchronous Data-Link Control
SDS	Software-Design Specification
SPEC	System Specification
UGV/TOV	Unmanned Ground Vehicle/Teleoperated Vehicle
VSI	Virtual Systems Interface
WAN	Wide Area Network

8.0 REFERENCES

Albus, J. S., H. G. McGain, and R. Lumina. 4 December 1984. NASA/NBS *Standard Reference Model for Telerobot Control System Architecture (NASREM)*.

Aviles, W. A., R. T., Laird, and M. E. Myers. November 1988. "Towards a Modular Robotic Architecture," *Proceedings SPIE Mobile Robots III*, pp. 271–278, Cambridge, MA.

Barbera, A. J., M. L. Fitzgerald, and J. S. Albus. April 1982. "Concepts for a Real-Time Sensory-Interactive Control System Architecture," *Proceedings of the Fourteenth Southeastern Symposium on System Theory*, pp. 121–126.

Barbera, A. J., M. L. Fitzgerald, J. S. Albus and L. S. Haynes. June 1984. "RCS: The NBS Real-Time Control System," presented at the *Robots 8 Conference and Exposition*, Detroit, MI.

Brendle, B. E., Jr. July 1990. "Robotic Vehicle Communications Interoperability Protocols," *Proceedings AUVS-90*, pp. 267–279, Dayton, OH.

Brooks, R. A. April 1986. "A Robust Layered Control System for a Mobile Robot," *IEEE Journal of Robotics and Automation*, RA-2, pp. 14–23, San Francisco, CA.

Everett, H. R., March 1988. "Security and Sentry Robots," *International Encyclopedia of Robotics Applications and Automation*, John Wiley & Sons, Inc., NY.

Everett, H. R., G. A. Gilbreath, T. T. Tran, and J. M. Nieusma. June 1990. *Modeling the Environment of a Mobile Security Robot*, NOSC Technical Document 1835, Naval Ocean Systems Center, San Diego, CA.

Hughes, T. W., H. R., Everett, A. Y. Umeda, S. W. Martin, W. A. Aviles, A. H. Koyomatsu, M. R. Solorzano, R. T. Laird, S. P. McArthur, and E. H. Spain. November 1990. "Issues in Mobile Robotics: Unmanned Ground Vehicle Program Teleoperated Vehicle," *Proceedings SPIE Mobile Robots V*, Boston, MA.

Intel Corporation. January 1989. "83C152 Hardware Description and Data Sheets," *Intel 8-Bit Embedded Controller Handbook*, No. 270645-002, pp. 9-1–9-87.

International Standards Organization (ISO). April 1980. *Reference Model for Open Systems Interconnection Architecture*, ISO/TC97/SC16 N309.

Lorin, H. 1988. *Aspects of Distributed Computer Systems, Second Edition*, John Wiley & Sons, Inc., NY.

McGain, H. G. March 1985. "A Hierarchically Controlled, Sensory Interactive Robot in the Automated Manufacturing Research Facility," *IEEE International Conference on Robotics and Automation*, pp. 931–939, St. Louis, MO.

Meystel, A. March 1988. "Mobile Robots, Autonomous," *International Encyclopedia of Robotics Applications and Automation*, John Wiley & Sons, Inc., NY.

APPENDIX A

SOFTWARE SYSTEM IMPLEMENTATION

Appendix A. Software System Implementation

The following section is a listing of the standard software subsystems of the MRA. The information provided herein is sufficient to implement application modules that can be configured to form a modular robot.

Configuration information in the form of directory and file specifications, as well as program build information (i.e., makefiles), are included in the listings.

The source for a single application module (Collision Avoidance Infrared - CAIR) is also included. This is an example of an object-oriented approach to the development of function-specific robot modules.

```
1     LD-List Directories, Advanced Edition 4.50, (C) Copr 1987-88, Peter Norton
2
3       C:\MRA
4       C:\MRA\APP
5       C:\MRA\APP\BIN
6       C:\MRA\APP\BIN\80152
7       C:\MRA\APP\BIN\8031
8       C:\MRA\APP\BIN\MSDOS
9       C:\MRA\APP\BIN\MSDOS\CS
10      C:\MRA\APP\BIN\MSDOS\WATCH
11      C:\MRA\APP\BIN\SBC8
12      C:\MRA\APP\SRC
13      C:\MRA\APP\SRC\AV
14      C:\MRA\APP\SRC\AV\TEST
15      C:\MRA\APP\SRC\CAIR
16      C:\MRA\APP\SRC\CAIR\TEST
17      C:\MRA\APP\SRC\CAUS
18      C:\MRA\APP\SRC\CAUS\TEST
19      C:\MRA\APP\SRC\CS
20      C:\MRA\APP\SRC\HLP
21      C:\MRA\APP\SRC\HLP\TEST
22      C:\MRA\APP\SRC\MOB
23      C:\MRA\APP\SRC\MOB\TEST
24      C:\MRA\APP\SRC\NAV
25      C:\MRA\APP\SRC\NAV\TEST
26      C:\MRA\APP\SRC\WATCH
27      C:\MRA\COM
28      C:\MRA\COM\BIN
29      C:\MRA\COM\BIN\80152
30      C:\MRA\COM\BIN\8031
31      C:\MRA\COM\BIN\MSDOS
32      C:\MRA\COM\BIN\SBC8
33      C:\MRA\COM\SRC
34      C:\MRA\COM\SRC\DEV
35      C:\MRA\COM\SRC\DEV\TEST
36      C:\MRA\COM\SRC\HDR
37      C:\MRA\COM\SRC\LCS
38      C:\MRA\COM\SRC\LCS\TEST
39      C:\MRA\COM\SRC\MMS
40      C:\MRA\COM\SRC\MMS\TEST
41      C:\MRA\ICN
42      C:\MRA\ICN\BIN
43      C:\MRA\ICN\BIN\80152
44      C:\MRA\ICN\BIN\80152\MON
45      C:\MRA\ICN\SRC
46      C:\MRA\ICN\SRC\AC
47      C:\MRA\ICN\SRC\AC\TEST
48      C:\MRA\ICN\SRC\GCS
49      C:\MRA\ICN\SRC\GCS\TEST
50      C:\MRA\LIB
51      C:\MRA\MPU
52      C:\MRA\MPU\BIN
53      C:\MRA\MPU\BIN\80152
54      C:\MRA\MPU\BIN\8031
55      C:\MRA\MPU\BIN\MSDOS
56      C:\MRA\MPU\BIN\SBC8
57      C:\MRA\MPU\SRC
58      C:\MRA\MPU\SRC\AC
59      C:\MRA\MPU\SRC\AC\TEST
60      C:\MRA\MPU\SRC\LDS
61      C:\MRA\MPU\SRC\LDS\TEST
62
63    59 directories
```

```
FI-File Info, Advanced Edition 4.50, (C) Copr 1987-88, Peter Norton

 Directory of C:\MRA

.               <DIR>      7-06-90   8:43a
..              <DIR>      7-06-90   8:43a
APP             <DIR>      8-07-90   2:24p
COM             <DIR>      7-06-90   8:43a
ICN             <DIR>      7-06-90   8:43a
LIB             <DIR>      5-30-91   8:05a
MPU             <DIR>      7-06-90   8:44a
makefile         11,243    5-30-91   1:00p
mradir            1,413    5-30-91  12:56p
mrafiles              0    5-30-91   1:01p

 Directory of C:\MRA\APP

.               <DIR>      8-07-90   2:24p
..              <DIR>      8-07-90   2:24p
BIN             <DIR>      8-07-90   2:24p
SRC             <DIR>      8-07-90   2:24p

 Directory of C:\MRA\APP\BIN

.               <DIR>      8-07-90   2:24p
..              <DIR>      8-07-90   2:24p
80152           <DIR>      3-25-91   3:34p
8031            <DIR>      3-25-91   2:19p
MSDOS           <DIR>      3-25-91   2:19p
SBC8            <DIR>      3-25-91   2:19p

 Directory of C:\MRA\APP\BIN\80152

.               <DIR>      3-25-91   3:34p
..              <DIR>      3-25-91   3:34p

 Directory of C:\MRA\APP\BIN\8031

.               <DIR>      3-25-91   2:19p
..              <DIR>      3-25-91   2:19p
cair              37,465   5-28-91  10:20a
caus              39,263   5-28-91  10:21a
mob               64,302   5-29-91  11:21a
cair     hex      52,624   5-28-91  10:21a
caus     hex      54,964   5-28-91  10:21a
mob      hex      87,366   5-29-91  11:21a
caird    obj       4,592   4-18-91   8:05a
cairl    obj       4,026   5-21-91   8:04a
causd    obj       6,347   5-08-91  10:11a
causl    obj       4,660   5-21-91   8:05a
fgm      obj         405   5-20-91   4:08p
irs      obj       6,934   5-24-91   4:25p
mobd     obj      17,755   5-27-91   1:26p
mobl     obj       8,264   5-27-91   1:26p
trc      obj      19,313   5-29-91  11:20a

 Directory of C:\MRA\APP\BIN\MSDOS

.               <DIR>      3-25-91   2:19p
..              <DIR>      3-25-91   2:19p
CS              <DIR>      4-05-91   2:50p
WATCH           <DIR>      4-05-91   8:22a
cairl    obj       3,461   5-21-91   8:04a
causl    obj       3,895   5-21-91   8:05a
mobl     obj       6,757   5-27-91   1:27p

 Directory of C:\MRA\APP\BIN\MSDOS\CS

.               <DIR>      4-05-91   2:50p
..              <DIR>      4-05-91   2:50p
cs       exe      28,061   5-29-91   7:43a
cs       map          31   5-08-91   7:40a
csd      obj       1,287   5-27-91   3:34p
```



```
main     obj       8,079   5-29-91   7:43a
mra      obj       1,020   4-16-91  10:49a

 Directory of C:\MRA\APP\BIN\MSDOS\WATCH

.               <DIR>      4-05-91   8:22a
..              <DIR>      4-05-91   8:22a
icnwatch exe     122,537   5-21-91   4:08p
icnwatch log       1,680   4-16-91  10:54a
main     obj      17,254   4-16-91  10:32a
mra      obj         809   4-16-91  10:32a

 Directory of C:\MRA\APP\BIN\SBC8

.               <DIR>      3-25-91   2:19p
..              <DIR>      3-25-91   2:19p

 Directory of C:\MRA\APP\SRC

.               <DIR>      8-07-90   2:24p
..              <DIR>      8-07-90   2:24p
AV              <DIR>      8-07-90   2:26p
CAIR            <DIR>      8-07-90   2:26p
CAUS            <DIR>      8-07-90   2:26p
CS              <DIR>      3-30-91   1:11p
HLP             <DIR>      8-07-90   2:26p
MOB             <DIR>      8-07-90   2:26p
NAV             <DIR>      8-07-90   2:26p
WATCH           <DIR>      4-05-91   8:23a

 Directory of C:\MRA\APP\SRC\AV

.               <DIR>      8-07-90   2:26p
..              <DIR>      8-07-90   2:26p
TEST            <DIR>      3-25-91   3:35p

 Directory of C:\MRA\APP\SRC\AV\TEST

.               <DIR>      3-25-91   3:35p
..              <DIR>      3-25-91   3:35p

 Directory of C:\MRA\APP\SRC\CAIR

.               <DIR>      8-07-90   2:26p
..              <DIR>      8-07-90   2:26p
TEST            <DIR>      3-06-91   3:41p
count                16    3-27-91   1:03p
makefile          5,620    5-30-91  12:43p
name                 16    3-27-91  12:43p
period1              28    4-08-91  11:54a
period15             28    4-05-91  10:33a
print                34    5-30-91  10:53a
range                16    3-27-91  12:47p
rate                 16    3-27-91   1:03p
report1              20    3-27-91   3:19p
reportn              24    3-27-91   3:19p
resetp               28    3-30-91   6:40p
resetw               24    4-05-91  10:39a
wait1                22    4-05-91  10:41a
wait5                22    4-08-91  11:56a
caird    31      55,948   4-18-91   8:05a
cairl    31      31,673   5-21-91   8:04a
cairl    at      11,315   5-21-91   8:04a
caird    c       20,426   4-18-91   7:45a
cairl    c        5,198   4-05-91   7:54a
caird    h        4,340   4-05-91   7:54a
cairl    h        2,953   5-16-91   4:50p
cair     m51    116,909   5-28-91  10:20a
cair     res        233   4-05-91   9:07a

 Directory of C:\MRA\APP\SRC\CAIR\TEST

.               <DIR>      3-06-91   3:41p
```

```
147   ..              <DIR>      3-06-91   3:41p
148
149    Directory of C:\MRA\APP\SRC\CAUS
150
151   .               <DIR>      8-07-90   2:26p
152   ..              <DIR>      8-07-90   2:26p
153   TEST            <DIR>      3-25-91   3:35p
154   count                16    4-18-91  10:23a
155   makefile          5,673    5-30-91  12:43p
156   name                 16    4-18-91  10:23a
157   period1              28    4-18-91  11:53a
158   period15             28    4-18-91  11:54a
159   period5              28    5-06-91  11:53a
160   print                34    5-30-91  10:53a
161   range                16    4-18-91  10:23a
162   rate                 16    4-18-91  10:26a
163   report1              20    4-18-91  10:39a
164   reportn              24    4-18-91  10:25a
165   resetp               28    4-18-91  11:56a
166   resetw               24    5-06-91   4:26p
167   setmin               24    5-08-91  10:30a
168   waitmin              22    5-06-91   4:28p
169   causd     31     72,652    5-08-91  10:11a
170   causl     31     36,079    5-21-91   8:05a
171   causl     at     12,820    5-21-91   8:05a
172   causd     c      26,784    5-08-91  10:07a
173   causl     c       5,605    4-18-91   4:22p
174   snr12     c       8,568    6-13-90  12:40p
175   sonar     c       9,609    4-10-91   7:46a
176   causd     h       4,525    4-18-91   3:58p
177   causl     h       3,205    5-16-91   4:50p
178   caus      m51   123,693    5-28-91  10:21a
179   caus      res       233    4-18-91   8:02a
180
181    Directory of C:\MRA\APP\SRC\CAUS\TEST
182
183   .               <DIR>      3-25-91   3:35p
184   ..              <DIR>      3-25-91   3:35p
185
186    Directory of C:\MRA\APP\SRC\CS
187
188   .               <DIR>      3-30-91   1:11p
189   ..              <DIR>      3-30-91   1:11p
190   makefile          6,539    5-30-91  12:45p
191   print                34    5-30-91  10:53a
192   csd       at      6,836    5-27-91   3:34p
193   main      at     24,094    5-29-91   7:43a
194   mra       at     11,442    4-16-91  10:49a
195   csd       c       3,789    5-27-91   3:34p
196   main      c      11,167    5-29-91   7:42a
197   mra       c       7,063    4-05-91   9:59a
198   csd       h       3,007    4-05-91   7:58a
199   mra       h       3,644    4-05-91   9:58a
200   cs        map    16,877    5-29-91   7:43a
201   cs        res       309    5-27-91   3:33p
202
203    Directory of C:\MRA\APP\SRC\HLP
204
205   .               <DIR>      8-07-90   2:26p
206   ..              <DIR>      8-07-90   2:26p
207   TEST            <DIR>      3-25-91   3:35p
208
209    Directory of C:\MRA\APP\SRC\HLP\TEST
210
211   .               <DIR>      3-25-91   3:35p
212   ..              <DIR>      3-25-91   3:35p
213
214    Directory of C:\MRA\APP\SRC\MOB
215
216   .               <DIR>      8-07-90   2:26p
217   ..              <DIR>      8-07-90   2:26p
218   TEST            <DIR>      8-09-90   4:52p
219   fwdjog               24    5-27-91  12:26p
```



```
220   fwdslow              24    5-27-91  12:17p
221   fwdstop              24    5-27-91  12:35p
222   halt                 18    5-27-91  12:15p
223   lftslow              24    5-27-91  12:52p
224   makefile          7,022    5-30-91  12:47p
225   name                 16    5-20-91   5:00p
226   print                34    5-30-91  10:53a
227   rate                 16    5-20-91   5:00p
228   reset                16    5-27-91   1:02p
229   revslow              24    5-27-91  12:53p
230   ritslow              24    5-27-91  12:52p
231   telemode             20    5-27-91  12:08p
232   fgm       31      8,482    5-20-91   4:08p
233   irs       31    109,106    5-24-91   4:25p
234   mobd      31    194,959    5-27-91   1:26p
235   mobl      31     65,153    5-27-91   1:26p
236   trc       31    238,803    5-29-91  11:20a
237   mobl      at     17,966    5-27-91   1:27p
238   fgm       c       5,262    5-20-91   4:08p
239   irs       c      40,778    5-20-91   4:24p
240   mobd      c      57,028    5-27-91   1:24p
241   mobl      c       8,596    5-27-91  11:27a
242   trc       c      90,951    5-29-91  11:18a
243   fgm       h       1,981    5-20-91   3:51p
244   irs       h       2,787    5-20-91  11:06a
245   mobd      h       6,177    5-27-91   1:24p
246   mobl      h       5,590    5-27-91   1:24p
247   trc       h       4,962    5-24-91   4:23p
248   mob       m51   209,381    5-29-91  11:21a
249   mob       res       322    5-16-91   4:38p
250
251    Directory of C:\MRA\APP\SRC\MOB\TEST
252
253   .               <DIR>      8-09-90   4:52p
254   ..              <DIR>      8-09-90   4:52p
255   trct             14,073    8-11-90   1:22p
256   c         bat        27    7-11-90   8:27a
257   l         bat        13    8-09-90   5:48p
258   m         bat        31    7-16-90   4:34p
259   o         bat        11    7-10-90   4:18p
260   trcold    c      71,926    4-05-91   9:19a
261   trct      c       1,457    4-05-91   8:18a
262   trcold    h       4,189    4-05-91   8:16a
263   trct      hex    13,519    8-11-90   1:22p
264   trct      lst    10,414    8-11-90   1:22p
265   trct      m51    55,294    8-11-90   1:22p
266   trct      obj     1,509    8-11-90   1:22p
267   trct      res       145    8-09-90   5:48p
268
269    Directory of C:\MRA\APP\SRC\NAV
270
271   .               <DIR>      8-07-90   2:26p
272   ..              <DIR>      8-07-90   2:26p
273   TEST            <DIR>      3-25-91   3:35p
274
275    Directory of C:\MRA\APP\SRC\NAV\TEST
276
277   .               <DIR>      3-25-91   3:35p
278   ..              <DIR>      3-25-91   3:35p
279
280    Directory of C:\MRA\APP\SRC\WATCH
281
282   .               <DIR>      4-05-91   8:23a
283   ..              <DIR>      4-05-91   8:23a
284   makefile          5,270    5-30-91  12:48p
285   print                34    5-30-91  10:53a
286   main      at     69,316    4-16-91  10:32a
287   mra       at     11,310    4-16-91  10:32a
288   main      c      36,559    4-09-91   2:35a
289   mra       c       7,063    4-05-91   8:48a
290   mra       h       3,644    4-05-91   8:46a
291   icnwatch  map    35,491    5-21-91   4:07p
292   icnwatch  res       177    4-05-91   9:11a
```

```
293
294    Directory of C:\MRA\COM
295
296    .                <DIR>      7-06-90   8:43a
297    ..               <DIR>      7-06-90   8:43a
298    BIN              <DIR>      7-06-90   8:45a
299    SRC              <DIR>      7-06-90   8:45a
300
301    Directory of C:\MRA\COM\BIN
302
303    .                <DIR>      7-06-90   8:45a
304    ..               <DIR>      7-06-90   8:45a
305    80152            <DIR>      3-20-91  11:58a
306    8031             <DIR>      3-20-91  11:56a
307    MSDOS            <DIR>      3-20-91  11:56a
308    SBC8             <DIR>      3-20-91  12:18p
309
310    Directory of C:\MRA\COM\BIN\80152
311
312    .                <DIR>      3-20-91  11:58a
313    ..               <DIR>      3-20-91  11:58a
314    lcd       obj     15,630    5-21-91   3:59p
315    lci       obj      1,755    3-28-91   3:53p
316    lmm       obj      5,753    4-08-91  10:29a
317    lmmdct    obj      1,299    4-16-91  10:13a
318    mm        obj     28,905    5-08-91  11:03a
319    pb        obj      3,855    5-28-91   8:25a
320    rtc       obj      2,077    4-01-91   9:47a
321    sio       obj      4,829    5-29-91   8:03a
322    smm       obj     12,199    4-08-91   2:08p
323    smmlib    obj      1,986    4-16-91  10:13a
324
325    Directory of C:\MRA\COM\BIN\8031
326
327    .                <DIR>      3-20-91  11:56a
328    ..               <DIR>      3-20-91  11:56a
329    lcd       obj     14,989    5-21-91   4:00p
330    lci       obj      1,755    3-28-91   3:54p
331    lmm       obj      5,753    4-08-91  10:36a
332    lmmdct    obj      1,299    4-16-91  10:15a
333    mm        obj     28,905    5-08-91  11:05a
334    pb        obj      3,855    5-28-91   8:25a
335    rtc       obj      2,077    4-01-91   9:47a
336    sio       obj      6,821    5-29-91   8:04a
337    smm       obj     12,199    4-08-91   2:09p
338    smmlib    obj      1,986    4-16-91  10:16a
339
340    Directory of C:\MRA\COM\BIN\MSDOS
341
342    .                <DIR>      3-20-91  11:56a
343    ..               <DIR>      3-20-91  11:56a
344    lcd       obj     15,143    5-21-91   4:01p
345    lci       obj      1,740    3-28-91   3:56p
346    lmm       obj      4,119    4-08-91  10:39a
347    lmmdct    obj      1,641    4-16-91  10:17a
348    mm        obj     17,349    5-08-91  11:06a
349    pb        obj      3,387    5-28-91   8:25a
350    rtc       obj      1,919    4-01-91   9:48a
351    sio       obj      3,611    5-29-91   8:05a
352    smm       obj      9,243    4-08-91   2:10p
353    smmlib    obj      2,194    4-16-91  10:18a
354
355    Directory of C:\MRA\COM\BIN\SBC8
356
357    .                <DIR>      3-20-91  12:18p
358    ..               <DIR>      3-20-91  12:18p
359    sio       obj      3,811    5-29-91   8:05a
360
361    Directory of C:\MRA\COM\SRC
362
363    .                <DIR>      7-06-90   8:45a
364    ..               <DIR>      7-06-90   8:45a
365    DEV              <DIR>      8-08-90   9:19a
```



```
366    HDR              <DIR>      7-08-90   4:50p
367    LCS              <DIR>     10-10-90  10:04a
368    MMS              <DIR>      7-06-90  12:28p
369
370    Directory of C:\MRA\COM\SRC\DEV
371
372    .                <DIR>      8-08-90   9:19a
373    ..               <DIR>      8-08-90   9:19a
374    TEST             <DIR>      8-08-90   3:18p
375    lib                  30     5-29-91   8:06a
376    makefile          5,434     5-30-91  12:35p
377    print                34     5-30-91  10:53a
378    rtc       152    31,021    4-01-91   9:47a
379    sio       152    97,153    5-29-91   8:03a
380    rtc       31     31,017    4-01-91   9:47a
381    sio       31    121,656    5-29-91   8:04a
382    rtc       at     20,997    4-01-91   9:48a
383    sio       at     54,177    5-29-91   8:05a
384    rtc       c      13,326    4-01-91   9:47a
385    sio       c      37,704    5-29-91   8:03a
386    rtc       h       1,764    3-28-91   9:11a
387    sio       h       5,455    5-15-91   8:19a
388    sio       sbc    54,375    5-29-91   8:05a
389
390    Directory of C:\MRA\COM\SRC\DEV\TEST
391
392    .                <DIR>      8-08-90   3:18p
393    ..               <DIR>      8-08-90   3:18p
394    siortc           10,098     5-29-91   1:27p
395    siotst            3,561     5-15-91   1:37p
396    c         bat        34     3-28-91   9:52a
397    l         bat        87     3-28-91   9:54a
398    o         bat        11     7-10-90   4:18p
399    siortc    c       1,510     5-29-91   1:26p
400    siotst    c         408     5-15-91   1:37p
401    siortc    exe    10,635     3-28-91   9:45a
402    siotst    exe     3,273     3-22-91  12:08p
403    siortc    hex    14,691     5-29-91   1:27p
404    siotst    hex     3,770     5-15-91   1:37p
405    siortc    lst    25,476     5-29-91   1:26p
406    siotst    lst     5,554     5-15-91   1:37p
407    siortc    m51    31,395     5-29-91   1:27p
408    siotst    m51    14,948     5-15-91   1:37p
409    siortc    obj     3,013     5-29-91   1:26p
410    siotst    obj       785     5-15-91   1:37p
411
412    Directory of C:\MRA\COM\SRC\HDR
413
414    .                <DIR>      7-08-90   4:50p
415    ..               <DIR>      7-08-90   4:50p
416    debug     h       1,431     3-22-91   1:47p
417    sysdefs   h       1,606     3-28-91   3:42p
418
419    Directory of C:\MRA\COM\SRC\LCS
420
421    .                <DIR>     10-10-90  10:04a
422    ..               <DIR>     10-10-90  10:04a
423    TEST             <DIR>     10-10-90  10:04a
424    lib                  30     5-21-91   4:02p
425    makefile          5,412     5-30-91  12:35p
426    print                34     5-30-91  10:53a
427    lcd       152   166,560    5-21-91   3:59p
428    lci       152    25,914    3-28-91   3:53p
429    lcd       31    159,933    5-21-91   4:00p
430    lci       31     25,910    3-28-91   3:54p
431    lcd       at     39,263    5-21-91   4:01p
432    lci       at     17,709    3-28-91   3:56p
433    bmf       c      26,091    3-22-91   3:56p
434    lcc       c      61,544    5-21-91   3:57p
435    lcd       c      23,403    3-28-91  12:14p
436    lci       c      10,848    3-22-91   3:56p
437    bmf       h       3,063    12-05-90  10:48a
438    lcc       h       5,722    3-28-91   4:24p
```

```
 lcd        h          3,745    2-01-91    3:33p
 lcl        h          3,209    3-26-91    4:00p

  Directory of C:\MRA\COM\SRC\LCS\TEST

 .                   <DIR>      10-10-90   10:04a
 ..                  <DIR>      10-10-90   10:04a
 packet             12,946      8-28-90    8:44a
 c          bat         38      8-28-90    2:24p
 l          bat         20      8-21-90    2:37p
 o          bat         11      7-10-90    4:18p
 packet     c          490      1-31-91    9:16a
 packet     hex     15,507      8-28-90    8:44a
 packet     lst      6,482      8-28-90    8:43a
 packet     m51     45,605      8-28-90    8:44a
 packet     obj      1,249      8-28-90    8:43a
 packet     res        186      8-23-90    4:46p

  Directory of C:\MRA\COM\SRC\MMS

 .                   <DIR>      7-06-90    12:28p
 ..                  <DIR>      7-06-90    12:28p
 TEST                <DIR>      10-26-90    9:53a
 lib                    30      5-28-91     8:28a
 makefile            7,295      5-30-91    12:36p
 print                  34      5-30-91    10:53a
 lmmdct     152     11,332      4-16-91    10:12a
 mm         152    280,122      5-08-91    11:03a
 pb         152     49,173      5-28-91     8:25a
 smmlib     152     17,995      4-16-91    10:13a
 lmmdct     31      11,328      4-16-91    10:15a
 mm         31     280,118      5-08-91    11:05a
 pb         31      49,169      5-28-91     8:25a
 smmlib     31      17,991      4-16-91    10:16a
 lmmdct     at       4,948      4-16-91    10:17a
 mm         at      65,321      5-08-91    11:06a
 pb         at      20,712      5-28-91     8:25a
 smmlib     at       7,678      4-16-91    10:18a
 lmm        c       19,952      4-16-91    10:07a
 lmmdct     c        2,584      3-28-91    12:12p
 mm         c       37,764      5-08-91    11:00a
 pb         c       11,843      5-28-91     8:23a
 smm        c       39,448      4-16-91    10:08a
 smmlib     c        3,292      4-08-91     9:06a
 lmm        h        4,567      3-28-91    12:37p
 mm         h        5,851      4-16-91     9:56a
 pb         h        2,341      4-05-91     9:27a
 smm        h        4,218      4-16-91     9:58a

  Directory of C:\MRA\COM\SRC\MMS\TEST

 .                   <DIR>      10-26-90    9:53a
 ..                  <DIR>      10-26-90    9:53a
 list                9,907      2-15-91    10:52a
 test               42,383      4-01-91     4:12p
 c          bat         23      1-10-91     8:23a
 l          bat         14      2-14-91     9:02a
 o          bat         11      7-10-90     4:18p
 list       c        5,527      2-15-91    10:54a
 test       c        1,915      5-08-91    10:52a
 test       exe     22,551      5-08-91    11:11a
 test       hex     53,424      1-24-91     4:31p
 list       lst     91,206      2-15-91    10:52a
 test       lst     36,310      4-01-91     4:07p
 list       m51     26,704      2-15-91    10:52a
 test       m51    116,262      4-01-91     4:12p
 list       obj      9,926      2-15-91    10:52a
 test       obj      1,666      5-08-91    11:11a
 list       res         50      2-14-91     9:04a
 test       res        116      4-01-91     4:11p

  Directory of C:\MRA\ICN
```



```
 .                   <DIR>      7-06-90    8:43a
 ..                  <DIR>      7-06-90    8:43a
 BIN                 <DIR>      7-06-90    8:47a
 SRC                 <DIR>      7-06-90    8:43a

  Directory of C:\MRA\ICN\BIN

 .                   <DIR>      7-06-90    8:47a
 ..                  <DIR>      7-06-90    8:47a
 80152               <DIR>      3-25-91   11:33a

  Directory of C:\MRA\ICN\BIN\80152

 .                   <DIR>      3-25-91   11:33a
 ..                  <DIR>      3-25-91   11:33a
 MON                 <DIR>      3-25-91   12:28p
 iac                 22,038     5-21-91    4:03p
 iac        hex      27,405     5-21-91    4:04p
 gcd        obj      13,781     3-29-91    1:16p
 gci        obj       1,772     3-28-91    4:13p
 main       obj         220     4-16-91   10:26a
 mra        obj       1,331     4-16-91   10:26a

  Directory of C:\MRA\ICN\BIN\80152\MON

 .                   <DIR>      3-25-91   12:28p
 ..                  <DIR>      3-25-91   12:28p
 icnmon              22,039     5-21-91    4:04p
 icnmon     hex      27,407     5-21-91    4:04p
 gcd        obj      13,782     3-29-91    1:17p

  Directory of C:\MRA\ICN\SRC

 .                   <DIR>      7-06-90    8:43a
 ..                  <DIR>      7-06-90    8:43a
 AC                  <DIR>      10-10-90   10:01a
 GCS                 <DIR>      7-06-90    8:43a

  Directory of C:\MRA\ICN\SRC\AC

 .                   <DIR>      10-10-90   10:01a
 ..                  <DIR>      10-10-90   10:01a
 TEST                <DIR>      11-01-90    8:13a
 makefile            6,332      5-30-91   12:50p
 print                  34      5-30-91   10:53a
 main       152      5,614      4-16-91   10:26a
 mra        152     20,971      4-16-91   10:26a
 main       c        1,876      3-17-91   10:18a
 mra        c        7,879      4-05-91    8:49a
 mra        h        3,644      4-05-91    8:59a
 iac        m51     73,852      5-21-91    4:04p
 icnmon     m51     73,927      5-21-91    4:04p
 iac        res        241      3-25-91    1:08p
 icnmon     res        285      3-25-91    1:14p

  Directory of C:\MRA\ICN\SRC\AC\TEST

 .                   <DIR>      11-01-90    8:13a
 ..                  <DIR>      11-01-90    8:13a
 pktio               11,114     2-05-91   10:39a
 pktiol              11,047     2-05-91   10:40a
 c          bat          37     11-01-90    8:25a
 l          bat          19     11-01-90    8:27a
 ll         bat          20      2-01-91    2:37p
 o          bat          11      7-10-90    4:18p
 pktio      c           519      3-28-91    8:09a
 pktiol     c           359      2-05-91   10:39a
 pktio      hex      15,252      2-05-91   10:39a
 pktiol     hex      15,195      2-05-91   10:40a
 pktio      lst       5,141      2-05-91   10:39a
 pktiol     lst       3,925      2-05-91   10:39a
 pktio      m51      34,644      2-05-91   10:39a
 pktiol     m51      34,279      2-05-91   10:40a
```

```
pktio     obj         859   2-05-91   10:39a
pktiol    obj         736   2-05-91   10:39a
pktio     res         170  12-19-90    4:20p
pktiol    res         171   2-01-91    2:37p

 Directory of C:\MRA\ICN\SRC\GCS

.                   <DIR>     7-06-90    8:43a
..                  <DIR>     7-06-90    8:43a
TEST                <DIR>     8-14-90    8:49a
lib                    30     5-30-91   10:44a
makefile            4,856     5-30-91   12:52p
print                  34     5-30-91   10:53a
gcd       152     148,333     3-29-91    1:16p
gci       152      26,897     3-28-91    4:13p
bmf       c        18,110     3-27-91    8:01a
gcd       c        26,732     3-28-91   10:19a
gci       c        11,005     2-01-91    3:38p
gsc       c        53,696     3-29-91    1:14p
bmf       h         2,993     3-27-91    7:56a
gcd       h         3,842     1-30-91    4:51p
gci       h         3,209     3-26-91    4:06p
gsc       h         7,566     3-14-91    2:55p
gcd       mon     148,345     3-29-91    1:17p

 Directory of C:\MRA\ICN\SRC\GCS\TEST

.                   <DIR>     8-14-90    8:49a
..                  <DIR>     8-14-90    8:49a
c         bat          27     7-11-90    8:27a
l         bat          90     7-10-90    4:15p
o         bat          11     7-10-90    4:18p

 Directory of C:\MRA\LIB

.                   <DIR>     5-30-91    8:05a
..                  <DIR>     5-30-91    8:05a
mra_1521  lib       78,072    5-30-91   10:44a
mra_311   lib       63,581    5-30-91    9:10a
mra_mss   lib       49,747    5-30-91   10:42a
mra_sbc8  lib        5,141    5-30-91   10:42a

 Directory of C:\MRA\MPU

.                   <DIR>     7-06-90    8:44a
..                  <DIR>     7-06-90    8:44a
BIN                 <DIR>     7-06-90    8:46a
SRC                 <DIR>     7-06-90    8:46a

 Directory of C:\MRA\MPU\BIN

.                   <DIR>     7-06-90    8:46a
..                  <DIR>     7-06-90    8:46a
80152               <DIR>     3-25-91    3:35p
8031                <DIR>     3-21-91    8:35a
MSDOS               <DIR>     3-21-91    8:35a
SBC8                <DIR>     3-25-91    1:39p

 Directory of C:\MRA\MPU\BIN\80152

.                   <DIR>     3-25-91    3:35p
..                  <DIR>     3-25-91    3:35p
ldi       obj          296    5-30-91    9:09a

 Directory of C:\MRA\MPU\BIN\8031

.                   <DIR>     3-21-91    8:35a
..                  <DIR>     3-21-91    8:35a
ldi       obj          296    5-30-91    9:09a
main      obj          367    4-16-91   10:28a
mra       obj          622    4-16-91   10:28a

 Directory of C:\MRA\MPU\BIN\MSDOS
```



```

.                   <DIR>     3-21-91    8:35a
..                  <DIR>     3-21-91    8:35a
ldi       obj          530    5-30-91    9:09a
main      obj        1,124    4-16-91   10:28a
mra       obj        1,072    4-16-91   10:29a

 Directory of C:\MRA\MPU\BIN\SBC8

.                   <DIR>     3-25-91    1:39p
..                  <DIR>     3-25-91    1:39p
ldi       obj          530    5-30-91    9:09a

 Directory of C:\MRA\MPU\SRC

.                   <DIR>     7-06-90    8:46a
..                  <DIR>     7-06-90    8:46a
AC                  <DIR>    10-10-90   10:01a
LDS                 <DIR>     7-06-90   12:29p

 Directory of C:\MRA\MPU\SRC\AC

.                   <DIR>    10-10-90   10:01a
..                  <DIR>    10-10-90   10:01a
TEST                <DIR>    11-01-90    8:14a
makefile            5,205    5-30-91   12:54p
print                  34    5-30-91   10:53a
main      31         6,988    4-16-91   10:28a
mra       31        14,930    4-16-91   10:28a
main      at         4,139    4-16-91   10:28a
mra       at        11,599    4-16-91   10:29a
main      c          2,301    4-05-91    8:29a
mra       c          7,063    4-05-91    8:50a
mra       h          3,644    4-05-91    8:57a

 Directory of C:\MRA\MPU\SRC\AC\TEST

.                   <DIR>    11-01-90    8:14a
..                  <DIR>    11-01-90    8:14a

 Directory of C:\MRA\MPU\SRC\LDS

.                   <DIR>     7-06-90   12:29p
..                  <DIR>     7-06-90   12:29p
TEST                <DIR>    10-10-90    2:17p
lib                    30    5-30-91    9:10a
makefile            4,722    5-30-91   12:55p
print                  34    5-30-91   10:53a
ldi       152        6,512    5-30-91    9:09a
ldi       31         6,508    5-30-91    9:09a
ldi       at         6,153    5-30-91    9:09a
ldi       c          3,819    5-30-91    9:08a
ldi       h          2,090    3-25-91   11:47a
ldi       sbc        6,153    5-30-91    9:09a

 Directory of C:\MRA\MPU\SRC\LDS\TEST

.                   <DIR>    10-10-90    2:17p
..                  <DIR>    10-10-90    2:17p

538 files found   21,860,352 bytes free
```

```
#*****************************************************************************#
#                                  MAKEFILE                                   #
#*****************************************************************************#
#                                                                             #
# CPCI:          IED90-MRA-MAKEFILE-TXT-R1C0                                  #
#                                                                             #
# Description:   Makefile for the Modular Robotic Architecture (MRA).         #
#                Updates any/all of the programming modules that reside       #
#                under the individual systems and subsystems.                 #
#                                                                             #
#                Targets are available for the following systems/subsystems:  #
#                                                                             #
#                mra    - MRA subsystems (make all)                           #
#                                                                             #
#                dev    - COM Standard Hardware Device Driver Subsystem       #
#                lcs    - COM Local Communications Subsystem                  #
#                mms    - COM Method Manager Subsystem                        #
#                                                                             #
#                gcs    - ICN Global Communications Subsystem                 #
#                iac    - ICN Applications Controller                         #
#                                                                             #
#                lds    - MPU Logical Device Subsystem                        #
#                mac    - MPU Applications Controller                         #
#                                                                             #
#                av     - MPU Audio/Visual Module                             #
#                cair   - MPU Collision Avoidance IR Module                   #
#                caus   - MPU Collision Avoidance Ultrasonic Module           #
#                cs     - MPU Control Station Module                          #
#                hlp    - MPU High-Level Processing Module                    #
#                mob    - MPU Mobility Module                                 #
#                nav    - MPU Navigation Module                               #
#                watch  - MPU ICNwatch Module                                 #
#                                                                             #
#                Compile-time literal definitions and their meanings follow:  #
#                                                                             #
#                I8031      - MPU using Intel 8031 running at 11.0592 MHz     #
#                IBMAT      - MPU using IBM-AT compatible runnning at 12 MHz  #
#                SBC8       - MPU using STD LPM-SBC8 (V20) running at 8 MHz   #
#                I80152     - ICN using Intel 80152 running at 14.7456 MHz    #
#                ICNMON     - ICN (80152) network monitor program             #
#                DEBUG      - Conditional compilation of debug code           #
#                                                                             #
# Notes:         1) The dependency and production rules are included here.     #
#                2) Linkage parameters for RAM-based systems using the CP-31: #
#                   CODE SEGMENT  = 08000h                                    #
#                   XDATA SEGMENT = 0C000h                                    #
#                   STARTUP CODE  = \c51\cram.obj                             #
#                3) Linkage parameters for ROM-based systems using the CP-31: #
#                   CODE SEGMENT  = 00000h                                    #
#                   XDATA SEGMENT = 08000h                                    #
#                   STARTUP CODE  = \c51\crom.obj                             #
#                4) Linkage parameters for ROM-based systems using the ICN:   #
#                   CODE SEGMENT  = 00000h                                    #
#                   XDATA SEGMENT = 00000h                                    #
#                   STARTUP CODE  = \c51\crom.obj                             #
#                5) Source files for most subsystems are the same between     #
#                   compilers/target systems. Conditional compilation is used #
#                   to generate the binary files for each system. The binary  #
#                   files are stored according to target system in the        #
#                   appropriate (\bin) directory. This allows the source file #
#                   name to be maintained while separating the binaries.      #
#                                                                             #
#                   Target System       Processor       Binary Directory      #
#                   ------------------------------------------------------    #
#                   CP-31               8031            bin\8031              #
#                   CP-31/535           8031/80535      bin\8031              #
#                   ICN                 80C152          bin\80152             #
#                   IBM-PC/AT           8088/80286      bin\MSDOS             #
#                   LPM-SBC8            8088/V20        bin\SBC8              #
#                                                                             #
#                   a) The binary files for different applications targeted   #
#                      for the same system are stored in subdirectories in    #
#                      the binary directory. For example, the Global Device   #
```



```
#                      Driver (GCD) subsystem binary file for the ICN monitor #
#                      application is stored as icn\bin\80152\mon\gcd.obj.    #
#                   b) File suffixes are used to distinguish between various  #
#                      target system files with the same (root) source file   #
#                      name.                                                  #
#                                                                             #
#                   Target System       File Suffix     Example               #
#                   ------------------------------------------------------    #
#                   CP-31               .31             rtc.31  (listing)     #
#                   CP-31/535           .31             trc.31                #
#                   ICN                 .152            lcd.152               #
#                   IBM-PC/AT           .at             main.at               #
#                   LPM-SBC8            .sbc            sio.sbc               #
#                                                                             #
# Edit History: 07/07/90 - Written by Robin T. Laird.                         #
#               05/27/91 - Last modified by Robin T. Laird.                   #
#                                                                             #
#*****************************************************************************#


################################### RULES ###################################

.SUFFIXES : .hex .exe .obj .c .a51

.IGNORE   :

$ Control settings for Franklin 8031 development

CC          =c51
AS          =a51
LINK        =l51
OTOH        =ohs51
CFLAGS      =cd la db sb
ASFLAGS     =
LFLAGS      =
OFLAGS      =
STARTUP     =\c51\crom.obj
CODESEG     =00000h
XDATASEG    =00000h

# Control settings for Microsoft MS-DOS development

MSC         =cl
MSAS        =masm
MSLINK      =link
MSCFLAGS    =/AS /c /Oi /Zi /Od
MSASFLAGS   =
MSLNKFLGS   =/co
LOADLIBES   =

.c.obj :
        $(CC) $< $(CFLAGS)

.a51.obj :
        $(AS) $< $(ASFLAGS)

.obj.exe :
        $(LINK) $(STARTUP),$< TO $@ code ($(CODESEG)) xdata ($(XDATASEG)) ixref

.exe.hex :
        $(OTOH) $< $(OFLAGS)


################################## DEFINITIONS ##############################

# Project, system, and application level definitions

PROJ            = \src\mra
APPSYS          = app
COMSYS          = com
ICNSYS          = icn
```

```
147  MPUSYS          = mpu
148
149  MRALIB          = \$(PROJ)\lib                          # Library binaries
150
151  APPSRC          = \$(PROJ)\$(APPSYS)\src                # Source files
152  COMSRC          = \$(PROJ)\$(COMSYS)\src
153  ICNSRC          = \$(PROJ)\$(ICNSYS)\src
154  MPUSRC          = \$(PROJ)\$(MPUSYS)\src
155
156  APPBIN31        = \$(PROJ)\$(APPSYS)\bin\8031           # 8031 binaries
157  COMBIN31        = \$(PROJ)\$(COMSYS)\bin\8031
158  ICNBIN31        = \$(PROJ)\$(ICNSYS)\bin\8031
159  MPUBIN31        = \$(PROJ)\$(MPUSYS)\bin\8031
160
161  APPBIN152       = \$(PROJ)\$(APPSYS)\bin\80152          # 80152 binaries
162  COMBIN152       = \$(PROJ)\$(COMSYS)\bin\80152
163  ICNBIN152       = \$(PROJ)\$(ICNSYS)\bin\80152
164  MPUBIN152       = \$(PROJ)\$(MPUSYS)\bin\80152
165  ICNBIN152_MON   = \$(PROJ)\$(ICNSYS)\bin\80152\mon
166
167  APPBINMS        = \$(PROJ)\$(APPSYS)\bin\msdos          # MS-DOS binaries
168  COMBINMS        = \$(PROJ)\$(COMSYS)\bin\msdos
169  ICNBINMS        = \$(PROJ)\$(ICNSYS)\bin\msdos
170  MPUBINMS        = \$(PROJ)\$(MPUSYS)\bin\msdos
171  MPUBINMS_WATCH  = \$(PROJ)\$(MPUSYS)\bin\msdos\watch
172
173  APPBINSBC8      = \$(PROJ)\$(APPSYS)\bin\sbc8           # LPM-SBC8 binaries
174  COMBINSBC8      = \$(PROJ)\$(COMSYS)\bin\sbc8
175  ICNBINSBC8      = \$(PROJ)\$(ICNSYS)\bin\sbc8
176  MPUBINSBC8      = \$(PROJ)\$(MPUSYS)\bin\sbc8
177
178  # Common subsystem level source directories
179
180  DEVSRC          = $(COMSRC)\dev
181  HDRSRC          = $(COMSRC)\hdr
182  LCSSRC          = $(COMSRC)\lcs
183  MMSSRC          = $(COMSRC)\mms
184
185  # ICN subsystem level source directories
186
187  GCSSRC          = $(ICNSRC)\gcs
188  IACSRC          = $(ICNSRC)\ac
189
190  # MPU subsystem level source directories
191
192  LDSSRC          = $(MPUSRC)\lds
193  MACSRC          = $(MPUSRC)\ac
194
195  # Application subsystem level source directories
196
197  AVSRC           = $(APPSRC)\av
198  CAIRSRC         = $(APPSRC)\cair
199  CAUSSRC         = $(APPSRC)\caus
200  CSSRC           = $(APPSRC)\cs
201  HLPSRC          = $(APPSRC)\hlp
202  MOBSRC          = $(APPSRC)\mob
203  NAVSRC          = $(APPSRC)\nav
204  WATCHSRC        = $(APPSRC)\watch
205
206
207
208  ################################### TARGETS ###################################
209
210  # MRA system target (the whole thing)
211
212  mra     : dev lcs mms iac gcs mac lds av cair caus cs hlp mob nav watch
213            cd \mra
214
215
216  # Common system targets
217
218  dev     :
219            cd $(DEVSRC)
```



```
220            make
221            make lib
222
223  lcs     :
224            cd $(LCSSRC)
225            make
226            make lib
227
228  mms     :
229            cd $(MMSSRC)
230            make
231            make lib
232
233
234  # ICN system targets
235
236  iac     :
237            cd $(IACSRC)
238            make
239
240  gcs     :
241            cd $(GCSSRC)
242            make
243            make lib
244
245
246  # MPU system targets
247
248  mac     :
249            cd $(MACSRC)
250            make
251
252  lds     :
253            cd $(LDSSRC)
254            make
255            make lib
256
257
258  # Application system targets
259
260  av      :
261            cd $(AVSRC)
262            make
263
264  cair    :
265            cd $(CAIRSRC)
266            make
267
268  caus    :
269            cd $(CAUSSRC)
270            make
271
272  cs      :
273            cd $(CSSRC)
274            make
275
276  hlp     :
277            cd $(HLPSRC)
278            make
279
280  mob     :
281            cd $(MOBSRC)
282            make
283
284  nav     :
285            cd $(NAVSRC)
286            make
287
288  watch   :
289            cd $(WATCHSRC)
290            make
```

```
1   #*************************************************************************#
2   #                              MAKEFILE                                   #
3   #*************************************************************************#
4   #                                                                         #
5   # CPCI:          IED90-MRA-APP-CAIR-MAKEFILE-TXT-R0C0                     #
6   #                                                                         #
7   # Description:   Makefile for the Mobile Security Robot application.     #
8   #                Makes the Collision Avoidance IR module subsystems.      #
9   #                                                                         #
10  #                Targets are available for the following systems/subsystems: #
11  #                                                                         #
12  #                cair       - MPU Collision Avoidance IR Module           #
13  #                cair.hex   - MPU Collision Avoidance IR Program (8031)   #
14  #                print      - Print Collision Avoidance IR source files   #
15  #                                                                         #
16  # Notes:         1) The dependency and production rules are included here. #
17  #                2) See also \mra\makefile.                               #
18  #                                                                         #
19  # Edit History: 03/25/91 - Written by Robin T. Laird.                     #
20  #                                                                         #
21  #*************************************************************************#
22
23
24
25  ##################################### RULES #####################################
26
27  .SUFFIXES : .hex .exe .obj .c .a51
28
29  # Control settings for Franklin 8031 development
30
31  CC          =c51
32  AS          =a51
33  LINK        =l51
34  OTOH        =ohs51
35  CFLAGS      =cd la db sb
36  ASFLAGS     =
37  LFLAGS      =
38  OFLAGS      =
39  STARTUP     =\c51\crom.obj
40  CODESEG     =00000h
41  XDATASEG    =00000h
42
43  # Control settings for Microsoft MS-DOS development
44
45  MSC         =cl
46  MSAS        =masm
47  MSLINK      =link
48  MSCFLAGS    =/AL /c /Oi /Zi /Od
49  MSASFLAGS   =
50  MSLNKFLGS   =/co
51  LOADLIBES   =
52
53  .c.obj :
54          $(CC) $< $(CFLAGS)
55
56  .a51.obj :
57          $(AS) $< $(ASFLAGS)
58
59  .obj.exe :
60          $(LINK) $(STARTUP),$< TO $@ code ($(CODESEG)) xdata ($(XDATASEG)) ixref
61
62  .exe.hex :
63          $(OTOH) $< $(OFLAGS)
64
65
66
67  ################################## DEFINITIONS ##################################
68
69  # Project, system, and application level definitions
70
71  PROJ            = mra
72  APPSYS          = app
73  COMSYS          = com
```



```
74  ICNSYS          = icn
75  MPUSYS          = mpu
76
77  APPSRC          = \$(PROJ)\$(APPSYS)\src
78  COMSRC          = \$(PROJ)\$(COMSYS)\src
79  ICNSRC          = \$(PROJ)\$(ICNSYS)\src
80  MPUSRC          = \$(PROJ)\$(MPUSYS)\src
81
82  APPBIN31        = \$(PROJ)\$(APPSYS)\bin\8031
83  APPBIN152       = \$(PROJ)\$(APPSYS)\bin\80152
84  APPBINMS        = \$(PROJ)\$(APPSYS)\bin\msdos
85  APPBINSBC8      = \$(PROJ)\$(APPSYS)\bin\sbc8
86
87  COMBIN31        = \$(PROJ)\$(COMSYS)\bin\8031
88
89  MPUBIN31        = \$(PROJ)\$(MPUSYS)\bin\8031
90
91  # Common subsystem level source directories
92
93  DEVSRC          = $(COMSRC)\dev
94  HDRSRC          = $(COMSRC)\hdr
95  LCSSRC          = $(COMSRC)\lcs
96  MMSSRC          = $(COMSRC)\mms
97
98  # ICN subsystem level source directories
99
100 GCSSRC          = $(ICNSRC)\gcs
101 IACSRC          = $(ICNSRC)\ac
102
103 # MPU subsystem level source directories
104
105 LDSSRC          = $(MPUSRC)\lds
106 MACSRC          = $(MPUSRC)\ac
107
108 # Application subsystem level source directories
109
110 CAIRSRC         = $(APPSRC)\cair
111
112 # Common subsystem global include and compilation units
113
114 SYSDEFS         = $(HDRSRC)\sysdefs.h
115
116 # Application subsystem compilation units
117
118 CAIR            = $(APPBIN31)\caird.obj         $(APPBIN31)\cairl.obj   \
119                   $(APPBINMS)\cairl.obj         $(MPUBIN31)\main.obj    \
120                   $(MPUBIN31)\mra.obj           $(COMBIN31)\rtc.obj     \
121                   $(COMBIN31)\mm.obj            $(COMBIN31)\pb.obj      \
122                   $(COMBIN31)\lmm.obj           $(COMBIN31)\smm.obj     \
123                   $(COMBIN31)\lmmdct.obj        $(COMBIN31)\smmlib.obj  \
124                   $(COMBIN31)\lci.obj           $(COMBIN31)\lcd.obj
125
126
127
128 ################################## TARGETS ##################################
129
130 cair                : $(APPBIN31)\cair.hex
131
132 $(APPBIN31)\cair.hex : $(APPBIN31)\cair
133                        $(OTOH) $< hex($@)
134
135 $(APPBIN31)\cair    : $(CAIR)
136                        $(LINK) @$(CAIRSRC)\cair.res
137
138 print               : $(CAIR)
139                        %-a2ps -nf caird.h | post
140                        %-a2ps -nf caird.c | post
141                        %-a2ps -nf cairl.h | post
142                        %-a2ps -nf cairl.c | post
143                        touch print
144
145
146
```

```
147  ############################ CAIR SYSTEM DEPENDENCIES ############################
148
149  # Collision Avoidance IR dictionary system dependencies
150
151  $(APPBIN31)\caird.obj : $(SYSDEFS)                                          \
152                          $(MMSSRC)\mm.h                                      \
153                          $(MMSSRC)\lmm.h                                     \
154                          $(CAIRSRC)\caird.h                                  \
155                          $(CAIRSRC)\caird.c
156    $(CC) $(CAIRSRC)\$*.c $(CFLAGS) df(18031) pr($(CAIRSRC)\$*.31) oj($(APPBIN31)
157
158
159  # Collision Avoidance IR library system dependencies
160
161  CAIRL = $(SYSDEFS)              $(MMSSRC)\mm.h          $(MMSSRC)\smm.h \
162          $(CAIRSRC)\cairl.h      $(CAIRSRC)\cairl.c
163
164  $(APPBIN31)\cairl.obj : $(CAIRL)
165    $(CC) $(CAIRSRC)\$*.c $(CFLAGS) df(18031) pr($(CAIRSRC)\$*.31) oj($(APPBIN31)
166
167  $(APPBINMS)\cairl.obj : $(CAIRL)
168    $(MSC) $(MSCFLAGS) /DIBMAT /Fs$(CAIRSRC)\$*.at /Fo$(APPBINMS)\$* $(CAIRSRC)\$
```

```
1       \c51\crom.obj,                  &
2       \mra\app\bin\8031\caird.obj,    &
3       \mra\mpu\bin\8031\main.obj,     &
4       \mra\mpu\bin\8031\mra.obj,      &
5       \mra\lib\mra_31l.lib            &
6       to \mra\app\bin\8031\cair       &
7       pr(\mra\app\src\cair\cair.m51) co(00000h) xd(08000h) ix
```

```
/*****************************************************************************\
*                                CAIRD.H                                      *
*****************************************************************************
*                                                                           *
* CPCI:          IED90-MRA-APP-CAIR-CAIRD-H-R0C0                            *
*                                                                           *
* Description:   Collision Avoidance IR (CAIR) module dictionary.           *
*                Contains prototypes for module-specific functions.         *
*                Defines local method dictionary data structure.            *
*                Defines the number of available module functions.          *
*                                                                           *
*                Module CAIRD.H exports the following variables/functions:  *
*                                                                           *
*                Module LMM.H:                                              *
*                int lmm_num_funcs;                                         *
*                int *lmm_dictionary[]();                                   *
*                                                                           *
*                Module LMM.H:                                              *
*                char v_obj_class[];                                        *
*                char v_obj_superclass[]                                    *
*                char v_unit_name[];                                        *
*                int  v_unit_reset;                                         *
*                                                                           *
*                CAIR (QUERY_OPERATING_LIMITS):                             *
*                cair_max_update_rate();                                    *
*                cair_number_sensors();                                     *
*                cair_sensor_range();                                       *
*                                                                           *
*                CAIR (STATUS_REQUEST, PERIODIC_STATUS_REQUEST):            *
*                cair_report_sensor();                                      *
*                cair_ir1_ir2_report_sensor();                              *
*                cair_wait_delta_sense();                                   *
*                                                                           *
* Notes:         1) Dictionaries include methods (functions) for all        *
*                   component classes of an object extending back to        *
*                   the root object (OBJECT).                               *
*                2) This file is included by the Local Method Manager (LMM).*
*                3) Error codes must be consistent with those in CAIRL.H.   *
*                                                                           *
* Edit History: 01/04/91 - Written by Robin T. Laird.                       *
*                                                                           *
\*****************************************************************************/

#ifndef CAIRD_MODULE_CODE
#define CAIRD_MODULE_CODE       20000

/* Public Data Structures:                                                  */

/* Number of functions in dictionary (including component functions).       */
/* Local method manager (LMM) global variable is defined/initialized here.  */

#define CAIR_NUM_FUNCS          10
int lmm_num_funcs               = CAIR_NUM_FUNCS;

/* Object class instance variables and values for this module.              */

char v_obj_class[]              = OBJ_MODULE_CLASS;
char v_obj_superclass[]         = OBJ_OBJECT_CLASS;

/* Unit class instance variables and values for this module.                */

char v_unit_name[]              = "CAIR";
int  v_unit_reset               = UNIT_NOT_INIT;

/* Prototype declarations for the dictionary functions.                     */
/* All functions return an int value indicating success/failure.            */

int cair_max_update_rate(), cair_number_sensors();
int cair_sensor_range(), cair_report_sensor();
int cair_ir1_ir2_report_sensor();
int cair_wait_delta_sense();

/* Function error codes.
```



```
/* Error codes indicate source of function execution failure.               */

#define CAIR_BAD_SENSOR_NUMBER  1

/* Dictionary data structure (pointers to the various functions).           */
/* Object (obj_) and unit (unit_) class methods are defined in LMM.DCT.     */
/* The function d_unit_reset() is called first upon system start-up.        */

int (*lmm_dictionary[CAIR_NUM_FUNCS]) () =
      { d_obj_class,            d_obj_superclass,       d_unit_name,
        d_unit_reset,           cair_max_update_rate,   cair_number_sensors,
        cair_sensor_range,      cair_report_sensor,     cair_ir1_ir2_report_sen
        cair_wait_delta_sense
      };

#endif
```

```
1   /*****************************************************************************\
2   *                                  CAIRD.C                                    *
3   ******************************************* *****************************************
4   *                                                                             *
5   * CPCI:          IED90-MRA-APP-CAIR-CAIRD-C-R0C0                              *
6   *                                                                             *
7   * Description:   Collision Avoidance IR (CAIR) dictionary functions.          *
8   *                These functions are available to all modules in the system.  *
9   *                Implements the (dictionary) functions for the collision      *
10  *                avoidance IR module (cair_).                                 *
11  *                                                                             *
12  *                Parameters are passed to/from the functions via the standard *
13  *                I/O buffers (mm_stdin and mm_stdout). All local methods must *
14  *                return an integer value indicating success or failure as in  *
15  *                MM_COMMAND_EXECUTED or MM_COMMAND_EXECUTION_FAILURE. Also,   *
16  *                for commands that fail, the reason for failure must be put   *
17  *                in the output buffer as a byte value.                        *
18  *                                                                             *
19  *                Module CAIRD.C exports the following variables/functions:    *
20  *                                                                             *
21  *                reset();                                                     *
22  *                cair_max_update_rate();                                      *
23  *                cair_number_sensors();                                       *
24  *                cair_sensor_range();                                         *
25  *                cair_report_sensor();                                        *
26  *                cair_ir1_ir2_report_sensor();                                *
27  *                cair_wait_delta_sense();                                     *
28  *                                                                             *
29  * Notes:         1) The calling sequence for each function is listed below.   *
30  *                                                                             *
31  * Edit History: 01/10/91 - Written by Robin T. Laird.                         *
32  *                                                                             *
33  \*****************************************************************************/
34
35  #include <absacc.h>                     /* Absolute addressing defs.        */
36  #include <sysdefs.h>                    /* System constants and types.      */
37  #include <mm.h>                         /* Method manager.                  */
38  #include <lmm.h>                        /* Local method manager.            */
39  #include "caird.h"                      /* CAIR dictionary entries.         */
40
41  /* Global "instance variables".                                             */
42
43  static XDATA byte v_cair_max_update_rate= 10;   /* Hz.                      */
44  static XDATA byte v_cair_number_sensors = 11;   /* Discrete.                */
45  static XDATA int  v_cair_sensor_range   = 300;  /* Tenths of inches.        */
46
47  /* The IR sensors are connected to the 8255 of the 8031 (CP-31).            */
48  /* IR sensors 1-8 are in port A, sensor 1 at bit 0, sensor 8 at bit 7.      */
49  /* IR sensors 9-11 are in port C, sensor 9 at bit 0, sensor 11 at bit 2.    */
50
51  /* Connector continuity is wired to the remaining bits of ports B and C.    */
52  /* If a sensor is connected, port bit value is 1, otherwise 0.              */
53  /* Continuity for IRs 1-8 are in port B, cont 1 at bit 0, cont 8 at bit 7.  */
54  /* Continuity for IRs 9-11 are in port C, cont 9 at bit 4, cont 11 at bit 6.*/
55
56  /* Addressing of the CP-31 puts 8255 ports A, B, and C in external data.    */
57
58  #define PA_8255         XBYTE[0x0000]   /* 8255 port A.                     */
59  #define PB_8255         XBYTE[0x0001]   /* 8255 port B.                     */
60  #define PC_8255         XBYTE[0x0002]   /* 8255 port C.                     */
61  #define PORTCON_8255    XBYTE[0x0003]   /* 8255 control register.           */
62
63  /* 8255 control values (see Intel books).                                   */
64
65  #define ALL_INPUT       0x9B            /* Ports A, B, and C as input.      */
66
67  /* Continuity check bit shift values and masks.                             */
68
69  #define PC_CONT_SHIFT   4               /* Shift to make a word.            */
70  #define PC_CONT_MASK    0x0070          /* Keep bits 4, 5, and 6.           */
71  #define PC_CONT_UNUSED  0x07FF          /* Masks out missing sensor bits.   */
72
73  /* Continuity check mask global variable.                                   */
```



```
74  /* Used to mask out (force to 0=OFF) all disconnected sensors.              */
75
76  static XDATA word cont_mask;
77
78
79  /*****************************************************************************\
80  *                                    reset                                    *
81  ******************************************************************************
82  *                                                                            *
83  * Function:      Resets (initializes) the CAIR.C module.                     *
84  *                All "housekeeping" required to initialize the module        *
85  *                is done here. This routine is called upon system power-up,  *
86  *                before any of the other module functions are called.        *
87  *                                                                            *
88  * Input:         reset();                                                    *
89  *                                                                            *
90  * Output:        Command execution status as an integer value (see MM.H).    *
91  *                                                                            *
92  * Globals:       cont_mask : module CAIR.C                                   *
93  *                                                                            *
94  * Edit History: 03/06/91 - Written by Robin T. Laird.                        *
95  *                                                                            *
96  \****************************************************************************/
97
98  int reset()
99  {
100   /* Set up the CP-31 8255 for input on ports A, B, and C.                   */
101
102   PORTCON_8255 = ALL_INPUT;
103
104   /* Check the IR sensor plug continuity, and set global sensor mask.       */
105   /* Complement mask since 0 = installed, 1 = not-installed.                */
106
107   cont_mask = (((word)PC_8255 & PC_CONT_MASK) << PC_CONT_SHIFT) | (word)PB_8255
108   cont_mask = PC_CONT_UNUSED & ~cont_mask;
109 }
110
111
112 /*****************************************************************************\
113 *                              cair_max_update_rate                           *
114 ******************************************************************************
115 *                                                                            *
116 * Function:      Returns the maximum update rate for the IR sensor report    *
117 *                functions (in Hz or updates per second). The value is       *
118 *                returned in the standard parameter output buffer.           *
119 *                                                                            *
120 * Input:         cair_max_update_rate();                                     *
121 *                                                                            *
122 * Output:        Max update rate as a byte value (mm_stdout).                *
123 *                                                                            *
124 * Globals:       mm_stdout : module MM.C                                     *
125 *                v_cair_max_update_rate : module CAIR.C                      *
126 *                                                                            *
127 * Edit History: 03/06/91 - Written by Robin T. Laird.                        *
128 *                                                                            *
129 \****************************************************************************/
130
131 int cair_max_update_rate()
132 {
133   /* Put max update rate in the standard parameter output buffer.           */
134   /* Return with command exec OK.                                           */
135
136   mm_sprintfb(&mm_stdout, "%b", v_cair_max_update_rate);
137   return(MM_COMMAND_EXECUTED);
138 }
139
140
141 /*****************************************************************************\
142 *                              cair_number_sensors                            *
143 ******************************************************************************
144 *                                                                            *
145 * Function:      Returns the number of IR sensors available.                 *
146 *                The value is returned in the standard parameter output      *
```

```
147   *                buffer.                                                *
148   *                                                                       *
149   * Input:         cair_number_sensors();                                 *
150   *                                                                       *
151   * Output:        Number of IR sensors as a byte value (mm_stdout).      *
152   *                                                                       *
153   * Globals:       mm_stdout : module MM.C                                *
154   *                v_cair_number_sensors : module CAIR.C                  *
155   *                                                                       *
156   * Edit History: 03/06/91 - Written by Robin T. Laird.                   *
157   *                                                                       *
158   \***********************     ***********************************************/
159
160   int cair_number_sensors
161   {
162     /* Put number of sensors in the standard parameter output buffer.     */
163     /* Return with command exec OK.                                       */
164
165     mm_sprintfb(&mm_stdout, "%b", v_cair_number_sensors);
166     return(MM_COMMAND_EXECUTED);
167   }
168
169
170   /*************************************************************************\
171   *                              cair_sensor_range                          *
172   *************************************************************************
173   *                                                                       *
174   * Function:      Returns the maximum range of the IR sensor (in tenths of *
175   *                inches). The value is returned in the standard parameter *
176   *                output buffer.                                         *
177   *                                                                       *
178   * Input:         cair_sensor_range();                                   *
179   *                                                                       *
180   * Output:        IR sensor range as an integer value (mm_stdout).       *
181   *                                                                       *
182   * Globals:       mm_stdout : module MM.C                                *
183   *                v_cair_sensor_range : module CAIR.C                    *
184   *                                                                       *
185   * Edit History: 03/06/91 - Written by Robin T. Laird.                   *
186   *                                                                       *
187   \*************************************************************************/
188
189   int cair_sensor_range()
190   {
191     /* Put sensor range in the standard parameter output buffer.          */
192     /* Return with command exec OK.                                       */
193
194     mm_sprintfb(&mm_stdout, "%d", v_cair_sensor_range);
195     return(MM_COMMAND_EXECUTED);
196   }
197
198
199   /*************************************************************************\
200   *                              cair_report_sensor                         *
201   *************************************************************************
202   *                                                                       *
203   * Function:      Reports the state (0=OFF, 1=ON) of the indicated sensor. *
204   *                The number of the sensor to report is passed in the    *
205   *                standard parameter input buffer as a bit value.        *
206   *                If the sensor number is invalid, an error code reporting so *
207   *                is returned via the standard parameter output buffer.  *
208   *                                                                       *
209   *                The sensors are attached to ports A, B, and C of the 8255. *
210   *                IR sensors 1-8 are in port A, IR1 at bit 0, IR8 at bit 7. *
211   *                IR sensors 9-11 are in port C, IR9 at bit 0, IR11 at bit 2. *
212   *                                                                       *
213   * Input:         cair_report_sensor(                                    *
214   *                  byte s; number of the sensor to report (mm_stdin).   *
215   *                );                                                     *
216   *                                                                       *
217   * Output:        State of given sensor (0/1) as a bit value (mm_stdout). *
218   *                                                                       *
219   * Globals:       mm_stdin  : module MM.C                                *
```



```
220   *                mm_stdout : module MM.C                                *
221   *                cont_mask : module CAIR.C                              *
222   *                v_cair_number_sensors : module CAIR.C                  *
223   *                                                                       *
224   * Edit History: 05/24/90 - Original code written by Richard P. Smurlo.  *
225   *               03/06/91 - Dictionary implementation by Robin T. Laird. *
226   *                                                                       *
227   \*************************************************************************/
228
229   int cair_report_sensor()
230   {
231     byte s, v;
232
233     /* Get the sensor number from the standard parameter input buffer.    */
234
235     mm_sscanfb(&mm_stdin, "%b", &s);
236
237     /* Make sure requested sensor is in range.                            */
238
239     if (s > v_cair_number_sensors)
240     {
241       /* Report invalid sensor number, and command failed.                */
242
243       mm_sprintfb(&mm_stdout, "%b", CAIR_BAD_SENSOR_NUMBER);
244       return(MM_COMMAND_EXECUTION_FAILURE);
245     }
246     else
247     {
248       /* Get the sensor value (0=OFF/not activated, 1=ON/activated).      */
249       /* Continuity mask is used to force disconnected sensors to 0=OFF.  */
250
251       v = (((((word)PC_8255 << 8) | (word)PA_8255) & cont_mask) >> s-1) & 1;
252
253       /* Put sensor value in standard parameter output buffer.            */
254
255       mm_sprintfb(&mm_stdout, "%y", v);
256       return(MM_COMMAND_EXECUTED);
257     }
258   }
259
260
261   /*************************************************************************\
262   *                          cair_ir1_ir2_report_sensor                     *
263   *************************************************************************
264   *                                                                       *
265   * Function:      Returns the state (0=OFF, 1=ON) of the indicated sensors. *
266   *                The numbers of the sensors to report are passed in the *
267   *                standard parameter input buffer as two byte values.    *
268   *                The states of sensors s1 through s2 are returned as a  *
269   *                sequence of bits (length = |s1-s2|+1). First sensor number *
270   *                should be less than the last sensor number.            *
271   *                If either sensor number is invalid, an error code reporting *
272   *                so is returned via the standard parameter output buffer. *
273   *                                                                       *
274   *                The sensors are attached to ports A, B, and C of the 8255. *
275   *                IR sensors 1-8 are in port A, IR1 at bit 0, IR8 at bit 7. *
276   *                IR sensors 9-11 are in port C, IR9 at bit 0, IR11 at bit 2. *
277   *                                                                       *
278   * Input:         cair_ir1_ir2_report_sensor(                            *
279   *                  byte s1; first sensor to sample (s1 <= s2). (mm_stdin). *
280   *                  byte s2; last sensor to sample (mm_stdin).           *
281   *                );                                                     *
282   *                                                                       *
283   * Output:        States of given sensors (0/1) as a bit string (mm_stdout). *
284   *                                                                       *
285   * Globals:       mm_stdin  : module MM.C                                *
286   *                mm_stdout : module MM.C                                *
287   *                cont_mask : module CAIR.C                              *
288   *                v_cair_number_sensors : module CAIR.C                  *
289   *                                                                       *
290   * Edit History: 05/24/90 - Original code written by Richard P. Smurlo.  *
291   *               03/06/91 - Dictionary implementation by Robin T. Laird. *
292   *                                                                       *
```

```
293  \*****************************************************************************/
294
295  int cair_ir1 ir2_report_sensor()
296  {
297    byte s1, s2, stemp, v;
298    word val1, i;
299
300    /* Get the sensor numbers from the standard parameter input buffer.       */
301
302    mm_sscanfb(&mm_stdin, "%b%b", &s1, &s2);
303
304    /* Make sure requested sensors are in range.                              */
305
306    if (s1 > v_cair_number_sensors || s2 > v_cair_number_sensors)
307    {
308      /* Report invalid sensor range, and command failed.                     */
309
310      mm_sprintfb(&mm_stdout, "%b", CAIR_BAD_SENSOR_NUMBER);
311      return(MM_COMMAND_EXECUTION_FAILURE);
312    }
313    else
314    {
315      /* Make sure first sensor is less than last sensor.                     */
316
317      if (s1 > s2)
318      {
319        stemp = s1;
320        s1 = s2;
321        s2 = stemp;
322      }
323
324      /* Get all sensor values (0=OFF/not activated, 1=ON/activated).         */
325      /* Continuity mask is used to force disconnected sensors to 0=OFF.      */
326
327      val1 = (((word)PC_8255 << 8) | (word)PA_8255) & cont_mask;
328
329      /* Loop through the sensors.                                            */
330
331      for (i = s1; i <= s2; i++)
332      {
333        /* Extract particular bit value for this sensor.                      */
334
335        v = (val1 >> i-1) & 1;
336
337        /* Put sensor value in standard parameter output buffer.              */
338
339        mm_sprintfb(&mm_stdout, "%y", v);
340      }
341      return(MM_COMMAND_EXECUTED);
342    }
343  }
344
345
346  /*****************************************************************************\
347  *                          cair_wait_delta_sense                            *
348  ******************************************************************************
349  *                                                                            *
350  * Function:     Returns the new state of the IR sensors (all of them)        *
351  *               if a change in states is seen between function calls.        *
352  *               All sensors are checked for a sense change in any            *
353  *               one sensor. The values of all sensors is returned upon       *
354  *               detecting a change (so the calling program must              *
355  *               determine which sensors actually changed states).            *
356  *                                                                            *
357  * Input:        cair_wait_delta_sense();                                     *
358  *                                                                            *
359  * Output:       States of all sensors as a word value (mm_stdout).           *
360  *                                                                            *
361  * Globals:      mm_stdout : module MM.C                                      *
362  *               cont_mask : module CAIR.C                                    *
363  *                                                                            *
364  * Edit History: 05/24/90 - Original code written by Richard P. Smurlo.       *
365  *               03/06/91 - Dictionary implementation by Robin T. Laird.      *
```



```
366  *                                                                            *
367  \*****************************************************************************/
368
369  int cair_wait_delta_sense()
370  {
371    static word prev_state = 0xFF;
372    word        curr_state;
373
374    /* Get current state of sensors.                                          */
375
376    curr_state = (((word)PC_8255 << 8) | (word)PA_8255) & cont_mask;
377
378    /* If we've detected a change, return sensor values, response required.   */
379    /* Otherwise, indicate no response message required.                      */
380
381    if (prev_state != curr_state)
382    {
383      prev_state = curr_state;
384      mm_sprintfb(&mm_stdout, "%u", curr_state);
385      return(MM_COMMAND_EXECUTED);
386    }
387    else
388      return(MM_SUPPRESS_OUTPUT);
389  }
390
```

```
1   /*****************************************************************************\
2   *                                CAIRL.H                                      *
3   *****************************************************************************
4   *                                                                           *
5   * CPCI:          IED90-MRA-APP-CAIR-CAIRL-H-R0C0                            *
6   *                                                                           *
7   * Description:   Collision Avoidance IR (CAIR) library function include file. *
8   *                Contains function prototypes and literals (#defines) for the *
9   *                collision avoidance IR module (cair_).                     *
10  *                                                                           *
11  *                Module CAIRL.H exports the following variables/functions:  *
12  *                                                                           *
13  *                CAIR (QUERY_OPERATING_LIMITS):                             *
14  *                cair_max_update_rate();                                    *
15  *                cair_number_sensors();                                     *
16  *                cair_sensor_range();                                       *
17  *                                                                           *
18  *                CAIR (STATUS_REQUEST, PERIODIC_STATUS_REQUEST):            *
19  *                cair_report_sensor();                                      *
20  *                cair_ir1_ir2_report_sensor();                              *
21  *                cair_wait_delta_sense();                                   *
22  *                                                                           *
23  * Notes:         1) The calling sequence for each function is listed below. *
24  *                2) Error codes must be consistent with those in CAIRD.H.   *
25  *                                                                           *
26  * Edit History: 01/22/91 - Written by Robin T. Laird.                       *
27  *                                                                           *
28  \*****************************************************************************/
29
30  #ifndef CAIRL_MODULE_CODE
31  #define CAIRL_MODULE_CODE       20500
32
33  #include <smm.h>                        /* System method manager.          */
34
35  /* Public data structures:                                                  */
36
37  #define CAIR_UNIT_NAME          "CAIR"
38
39  #define CAIR_FN_MAX_UPDATE_RATE (0+SMM_NUM_INHERITED_FNS)
40  #define CAIR_FN_NUMBER_SENSORS  (1+SMM_NUM_INHERITED_FNS)
41  #define CAIR_FN_SENSOR_RANGE    (2+SMM_NUM_INHERITED_FNS)
42  #define CAIR_FN_REPORT_SENSOR   (3+SMM_NUM_INHERITED_FNS)
43  #define CAIR_FN_IR1_IR2_REPORT  (4+SMM_NUM_INHERITED_FNS)
44  #define CAIR_FN_WAIT_DELTA      (5+SMM_NUM_INHERITED_FNS)
45
46  /* Function error codes.
47  /* Error codes indicate source of function execution failure.              */
48
49  #define CAIR_BAD_SENSOR_NUMBER  1
50
51  /* Public functions:                                                        */
52
53  int cair_max_update_rate(void);
54  int cair_number_sensors(void);
55  int cair_sensor_range(void);
56  int cair_report_sensor(byte s, word period);
57  int cair_ir1_ir2_report_sensor(byte s1, byte s2, word period);
58  int cair_wait_delta_sense(word period);
59
60  #endif
```

```
/*****************************************************************************\
*                                  CAIRL.C                                    *
*****************************************************************************
*                                                                            *
* CPCI:          IED90-MRA-APP-CAIR-CAIRL-C-R0C0                             *
*                                                                            *
* Description:   Collision Avoidance IR (CAIR) library functions.            *
*                These functions are available to all modules in the system. *
*                Implements the (library) functions for the collision        *
*                avoidance IR module (cair_).                                *
*                                                                            *
*                Parameters are passed to/from the functions via the standard *
*                I/O buffers (mm_stdin and mm_stdout). All local methods must *
*                return an integer value indicating success or failure as in *
*                MM_COMMAND_EXECUTED or MM_COMMAND_EXECUTION_FAILURE. Also,  *
*                for commands tha fail, the reason for failure must be put    *
*                in the output buffer as a byte value.                       *
*                                                                            *
*                Module CAIRL.C exports the following variables/functions:   *
*                                                                            *
*                cair_max_update_rate();                                     *
*                cair_number_sensors();                                      *
*                cair_sensor_range();                                        *
*                cair_report_sensor();                                       *
*                cair_ir1_ir2_report_sensor();                               *
*                cair_wait_delta_sense();                                    *
*                                                                            *
* Notes:         1) The calling sequence for each function is listed below.  *
*                                                                            *
* Edit History: 01/10/91 - Written by Robin T. Laird.                        *
*                                                                            *
\*****************************************************************************/

#include <sysdefs.h>                    /* System constants and types.      */
#include <mm.h>                         /* Method manager.                  */
#include <smm.h>                        /* System method manager.           */
#include "cairl.h"                      /* Unit name and function IDs.      */


int cair_max_update_rate()
{
  int         event;
  mm_message m;

  m.trans_disposition = MM_INITIATING;
  m.trans_category    = MM_QUERY_OPERATING_LIMITS;
  m.function_id       = CAIR_FN_MAX_UPDATE_RATE;
  smm_generate_message(CAIR_UNIT_NAME, &m, &event);
  return(event);
}


int cair_number_sensors()
{
  int         event;
  mm_message m;

  m.trans_disposition = MM_INITIATING;
  m.trans_category    = MM_QUERY_OPERATING_LIMITS;
  m.function_id       = CAIR_FN_NUMBER_SENSORS;
  smm_generate_message(CAIR_UNIT_NAME, &m, &event);
  return(event);
}


int cair_sensor_range()
{
  int         event;
  mm_message m;

  m.trans_disposition = MM_INITIATING;
  m.trans_category    = MM_QUERY_OPERATING_LIMITS;
  m.function_id       = CAIR_FN_SENSOR_RANGE;
```



```
  smm_generate_message(CAIR_UNIT_NAME, &m, &event);
  return(event);
}


int cair_report_sensor(s, period)
byte s;
word period;
{
  int         event;
  mm_message m;

  m.trans_disposition = MM_INITIATING;
  m.function_id       = CAIR_FN_REPORT_SENSOR;

  /* If a periodic status request, then period will be non-NULL.            */

  if (period == NULL)
  {
    m.trans_category  = MM_STATUS_REQUEST;
    mm_sprintfb(&mm_stdout, "%b", s);
  }
  else
  {
    m.trans_category  = MM_PERIODIC_STATUS_REQUEST;
    mm_sprintfb(&mm_stdout, "%u%b", period, s);
  }
  smm_generate_message(CAIR_UNIT_NAME, &m, &event);
  return(event);
}


int cair_ir1_ir2_report_sensor(s1, s2, period)
byte s1, s2;
word period;
{
  int         event;
  mm_message m;

  m.trans_disposition = MM_INITIATING;
  m.function_id       = CAIR_FN_IR1_IR2_REPORT;

  /* If a periodic status request, then period will be non-NULL.            */

  if (period == NULL)
  {
    m.trans_category  = MM_STATUS_REQUEST;
    mm_sprintfb(&mm_stdout, "%b%b", s1, s2);
  }
  else
  {
    m.trans_category  = MM_PERIODIC_STATUS_REQUEST;
    mm_sprintfb(&mm_stdout, "%u%b%b", period, s1, s2);
  }
  smm_generate_message(CAIR_UNIT_NAME, &m, &event);
  return(event);
}


int cair_wait_delta_sense(period)
word period;
{
  int         event;
  mm_message m;

  m.trans_disposition = MM_INITIATING;
  m.function_id       = CAIR_FN_WAIT_DELTA;

  /* If a periodic status request, then period will be non-NULL.            */

  if (period == NULL)
  {
    m.trans_category  = MM_STATUS_REQUEST;
```

```
147     }
148     else
149     {
150       m.trans_category  = MM_PERIODIC_STATUS_REQUEST;
151       mm_sprintfb(&mm_stdout, "%u", period);
152     }
153     smm_generate_message(CAIR_UNIT_NAME, &m, &event);
154     return(event);
155   }
```

```
#*****************************************************************************#
#                                  MAKEFILE                                   #
#*****************************************************************************#
#                                                                             #
# CPCI:          IED90-MRA-COM-DEV-MAKEFILE-TXT-R0C0                          #
#                                                                             #
# Description:   Makefile for the Modular Robotic Architecture (MRA).         #
#                Makes the common device driver subsystem.                    #
#                                                                             #
#                Targets are available for the following systems/subsystems:  #
#                                                                             #
#                dev    - COM Standard Hardware Device Driver Subsystem       #
#                lib    - Add modules to MRA library                          #
#                print  - Print COM device driver files                       #
#                                                                             #
# Notes:         1) The dependency and production rules are included here.    #
#                2) See also \mra\makefile.                                   #
#                                                                             #
# Edit History: 03/22/91 - Written by Robin T. Laird.                         #
#                                                                             #
#*****************************************************************************#


###################################### RULES ######################################

.SUFFIXES : .hex .exe .obj .c .a51

# Control settings for Franklin 8031 development

CC          =c51
AS          =a51
LINK        =l51
OTOH        =ohs51
CFLAGS      =cd la db sb
ASFLAGS     =
LFLAGS      =
OFLAGS      =
STARTUP     =\c51\crom.obj
CODESEG     =00000h
XDATASEG    =00000h

# Control settings for Microsoft MS-DOS development

MSC         =cl
MSAS        =masm
MSLINK      =link
MSCFLAGS    =/AL /c /Oi /Zi /Od
MSASFLAGS   =
MSLNKFLGS   =/co
LOADLIBES   =

.c.obj :
        $(CC) $< $(CFLAGS)

.a51.obj :
        $(AS) $< $(ASFLAGS)

.obj.exe :
        $(LINK) $(STARTUP),$< TO $@ code ($(CODESEG)) xdata ($(XDATASEG)) ixref

.exe.hex :
        $(OTOH) $< $(OFLAGS)


###################################### DEFINITIONS ######################################

# Project, system, and application level definitions

PROJ            = mra
APPSYS          = app
COMSYS          = com
```



```

COMLIB          = \$(PROJ)\lib
COMSRC          = \$(PROJ)\$(COMSYS)\src
COMBIN31        = \$(PROJ)\$(COMSYS)\bin\8031
COMBIN152       = \$(PROJ)\$(COMSYS)\bin\80152
COMBINMS        = \$(PROJ)\$(COMSYS)\bin\msdos
COMBINSBC8      = \$(PROJ)\$(COMSYS)\bin\sbc8

# Common subsystem level source directories

DEVSRC          = $(COMSRC)\dev
HDRSRC          = $(COMSRC)\hdr

# Common subsystem global include and compilation units

SYSDEFS         = $(HDRSRC)\sysdefs.h

DEV             = $(COMBIN152)\rtc.obj          $(COMBIN31)\rtc.obj     \
                  $(COMBINMS)\rtc.obj           $(COMBIN152)\sio.obj    \
                  $(COMBIN31)\sio.obj           $(COMBINMS)\sio.obj     \
                  $(COMBINSBC8)\sio.obj


###################################### TARGETS ######################################

dev             : $(DEV)

lib             : $(DEV)
                  -lib51 delete $(COMLIB)\mra_1521.lib (rtc, sio)
                  -lib51 add $(COMBIN152)\rtc.obj to $(COMLIB)\mra_1521.lib
                  -lib51 add $(COMBIN152)\sio.obj to $(COMLIB)\mra_1521.lib
                  -lib51 delete $(COMLIB)\mra_311.lib (rtc, sio)
                  -lib51 add $(COMBIN31)\rtc.obj to $(COMLIB)\mra_311.lib
                  -lib51 add $(COMBIN31)\sio.obj to $(COMLIB)\mra_311.lib
                  -lib $(COMLIB)\mra_mss.lib -rtc.obj+$(COMBINMS)\rtc.obj;
                  -lib $(COMLIB)\mra_msl.lib -rtc.obj+$(COMBINMS)\rtc.obj;
                  -lib $(COMLIB)\mra_mss.lib -sio.obj+$(COMBINMS)\sio.obj;
                  -lib $(COMLIB)\mra_msl.lib -sio.obj+$(COMBINMS)\sio.obj;
                  -lib $(COMLIB)\mra_sbc8.lib -sio.obj+$(COMBINSBC8)\sio.obj;
                  touch lib

print:            $(DEV)
                  %-a2ps -nf rtc.h | post
                  %-a2ps -nf rtc.c | post
                  %-a2ps -nf sio.h | post
                  %-a2ps -nf sio.c | post
                  touch print


############################## COM DEV DEPENDENCIES ##############################

# COM Real Time Clock module dependencies for 80152, 8031, MS-DOS

RTC = $(SYSDEFS) $(DEVSRC)\rtc.h $(DEVSRC)\rtc.c

$(COMBIN152)\rtc.obj            : $(RTC)
  $(CC) $(DEVSRC)\6*.c $(CFLAGS) df(I80152) pr($(DEVSRC)\$*.152) oj($(COMBIN152

$(COMBIN31)\rtc.obj             : $(RTC)
  $(CC) $(DEVSRC)\$*.c $(CFLAGS) df(I8031)  pr($(DEVSRC)\$*.31)  oj($(COMBIN31)

$(COMBINMS)\rtc.obj             : $(RTC)
  $(MSC) $(MSCFLAGS) /DIBMAT /Fs$(DEVSRC)\$*.at /Fo$(COMBINMS)\$* $(DEVSRC)\$*.

# COM Serial I/O module dependencies 80152, 8031, MS-DOS

SIO = $(SYSDEFS) $(DEVSRC)\sio.h $(DEVSRC)\sio.c

$(COMBIN152)\sio.obj            : $(SIO)
  $(CC) $(DEVSRC)\$*.c $(CFLAGS) df(I80152) pr($(DEVSRC)\$*.152) oj($(COMBIN152
```

```
147
148   $(COMBIN31)\sio.obj             : $(SIO)
149     $(CC) $(DEVSRC)\$*.c $(CFLAGS) df(IB031)  pr($(DEVSRC)\$*.31)  oj($(COMBIN31)
150
151   $(COMBINMS)\sio.obj             : $(SIO)
152     $(MSC) $(MSCFLAGS) /DIBMAT /Fs$(DEVSRC)\$*.at /Fo$(COMBINMS)\$* $(DEVSRC)\$*.
153
154   $(COMBINSBC8)\sio.obj           : $(SIO)
155     $(MSC) $(MSCFLAGS) /DSBC8 /Fs$(DEVSRC)\$*.sbc /Fo$(COMBINSBC8)\$* $(DEVSRC)\$
```

```
1     /*****************************************************************************\
2     *                                    rtc.h                                    *
3     *****************************************************************************
4     *                                                                             *
5     * CPCI:          IED90-MRA-COM-DEV-RTC-H-R0C1                                 *
6     *                                                                             *
7     * Description:   Real-time clock device driver external declarations.         *
8     *                Contains the constant definitions and function prototypes    *
9     *                for the RTC.C module. These functions provide standard       *
10    *                access to the on-board real-time clock.                      *
11    *                                                                             *
12    *                Module RTC exports the following functions:                  *
13    *                                                                             *
14    *                rtc_init();                                                  *
15    *                rtc_wait();                                                  *
16    *                rtc_time();                                                  *
17    *                                                                             *
18    * Notes:         1) The RTC functions are part of the MRA standard            *
19    *                   hardware device driver (DEV) subsystem.                   *
20    *                                                                             *
21    * Edit History: 03/28/91 - Modified by Robin T. Laird.                        *
22    *                                                                             *
23    \*****************************************************************************/
24
25    /* Public Data Structures:                                                   */
26
27    #ifndef RTC_MODULE_CODE
28    #define RTC_MODULE_CODE          5000
29
30    #define ERR_RTC_NOT_INIT         1+RTC_MODULE_CODE
31
32    #define RTC_RANDOM_TIME          0L
33
34    /* External module global error variable.                                   */
35
36    extern int rtc_error;
37
38    /* Public Functions:                                                         */
39
40    void rtc_init(void);
41    void rtc_wait(unsigned long duration);
42    unsigned long rtc_time(void);
43
44    #endif
```

```
1   /*****************************************************************************\
2   *                                  rtc.c                                      *
3   ******************************************************************************
4   *                                                                            *
5   * CPCI:          IED90-MRA-COM-DEV-RTC-C-R0C1                                *
6   *                                                                            *
7   * Description:   Real-time clock device driver functions.                    *
8   *                Implements a real-time clock with approximate millesecond   *
9   *                resolution. The functions below are currently implemented   *
10  *                Currently implemented for the Intel 8031 microcontroller.   *
11  *                                                                            *
12  *                Module RTC exports the following functions:                 *
13  *                                                                            *
14  *                rtc_init();                                                 *
15  *                rtc_wait();                                                 *
16  *                rtc_time();                                                 *
17  *                                                                            *
18  * Notes:         1) The RTC functions are part of the MRA standard           *
19  *                   hardware device driver (DEV) subsystem.                  *
20  *                2) See the Intel 8-Bit Embedded Controllers handbook        *
21  *                   for more information (No. 270645-002, pp. 7-7 - 7-11).   *
22  *                                                                            *
23  * Edit History: 05/02/90 - Modified by Robin T. Laird.                       *
24  *                                                                            *
25  \*****************************************************************************/
26
27  #if defined(IBMAT)
28  #include <timeb.h>                      /* MS-DOS time struct definitions.  */
29  #else
30  #include <reg51.h>                      /* SFR definitions for 8031.        */
31  #endif
32  #include <stdlib.h>                     /* Random number generator fns.     */
33  #include <sysdefs.h>                    /* System constants and types.      */
34  #include <rtc.h>                        /* Real-time clock constants.       */
35
36  /* Public Variables:                                                         */
37
38  /* Global module error variable, rtc_error.                                 */
39  /* rtc_error contains code of last error occurence.                          */
40  /* Should be set to AOK after each successful function call.                 */
41  /* Variable can be examined by other software after each function call.      */
42
43  XDATA int rtc_error = ERR_RTC_NOT_INIT; /* Global module error variable.    */
44
45  /* Declare internal clock tick counter.                                      */
46  /* Rolls over after 4,294,967,295 counts.                                    */
47
48  #if defined(I80152) || defined(I8031)
49  static data unsigned long tickcnt = 0L;
50  #endif
51
52
53  /*****************************************************************************\
54  *                                rtc_tickint                                 *
55  ******************************************************************************
56  *                                                                            *
57  * Function:      Tick count interrupt routine. Increments the module         *
58  *                variable tickcnt every timer interrupt. Represents          *
59  *                elapsed time since the initialization routine was           *
60  *                last called. The rate at which the function is              *
61  *                called depends upon the TIMER/COUNTER mode and other        *
62  *                parameters specified in the initialization routine.         *
63  *                8031 family interrupt number 1 is TIMER/COUNTER-0.          *
64  *                Use register bank 2 for handling this interrupt.            *
65  *                                                                            *
66  * Input:         rtc_tickint();                                              *
67  *                                                                            *
68  * Output:        Nothing.                                                    *
69  *                                                                            *
70  * Globals:       tickcnt : module RTC.C                                      *
71  *                                                                            *
72  * Edit History: 06/19/90 - Written by Robin T. Laird.                        *
73  *                                                                            *
```



```
74  \*****************************************************************************/
75
76  #if !defined(IBMAT)
77  static void rtc_tickint() interrupt 1 using 2
78  {
79    tickcnt++;
80  }
81  #endif
82
83
84  /*****************************************************************************\
85  *                                  rtc_init                                  *
86  ******************************************************************************
87  *                                                                            *
88  * Function:      Routine to initialize the CP-31 (8031) TIMER 0 for          *
89  *                use as a real-time (pseudo-millisecond resoultion)          *
90  *                clock. TIMER 0 is configured to operate in mode 0           *
91  *                which is a 13-bit counter that interrupts the CPU           *
92  *                every OSCfrequency/(12*8192*1000) milliseconds.             *
93  *                This routine also clears the module tick count variable     *
94  *                tickcnt, and then starts the timer.                         *
95  *                                                                            *
96  * Input:         rtc_init();                                                 *
97  *                                                                            *
98  * Output:        Nothing.                                                    *
99  *                                                                            *
100 * Globals:       rtc_error : module RTC.C                                    *
101 *                tickcnt   : module RTC.C                                    *
102 *                                                                            *
103 * Edit History: 06/19/90 - Written by Robin T. Laird.                        *
104 *                03/28/91 - Modified by Robin T. Laird to init srand().      *
105 *                                                                            *
106 \*****************************************************************************/
107
108 /* Define 8031 timer/counter control register bit functions (for TIMER 0).  */
109
110 #define GATE_CONTROL            0x08
111 #define NO_GATE_CONTROL         0x00
112
113 #define TIMER_FUNCTION          0x00
114 #define COUNTER_FUNCTION        0x04
115
116 #define MODE_0                  0x00
117 #define MODE_1                  0x01
118 #define MODE_2                  0x02
119 #define MODE_3                  0x03
120
121 void rtc_init()
122 {
123   rtc_error = AOK;              /* Assume function successful...            */
124
125   #if !defined(IBMAT)
126
127   /* Clear TIMER 0 control bits.                                             */
128   /* Set TIMER 0 to MODE 0: 13-bit timer (MCS-48 compatible).                */
129
130   TMOD &= 0xF0;
131   TMOD |= NO_GATE_CONTROL | TIMER_FUNCTION | MODE_0;
132
133   /* Expicitly zero tick counter (one way of resetting clock).               */
134   /* Enable TIMER/COUNTER-0 interrupts.                                      */
135   /* Turn TIMER/COUNTER on (1=on, 0=off).                                    */
136
137   tickcnt = 0L;
138
139   ET0 = 1;                      /* TIMER 0 interrupt enable bit.            */
140   TR0 = 1;                      /* TIMER 0 run control bit.                 */
141   EA  = 1;                      /* Enable processor interrupts.             */
142
143   #endif
144
145   /* Seed random number generator used for wait function below.              */
146
```

```
147     srand((int)rtc_time());
148   }
149
150
151   /*************************************************************************\
152   *                                rtc_wait                                 *
153   *************************************************************************
154   *                                                                         *
155   * Function:     Waits for a specified period of time (in ms).             *
156   *               Function does not return until time expired.              *
157   *               If duration parameter is RTC_RANDOM_TIME then the function *
158   *               waits a random amount of time between 2000 and 7000 ms.   *
159   *                                                                         *
160   * Input:        rtc_wait(                                                 *
161   *                 unsigned long duration; time to wait or RTC_RANDOM_TIME. *
162   *               );                                                        *
163   *                                                                         *
164   * Output:       Nothing.                                                  *
165   *                                                                         *
166   * Globals:      rtc_error : module RTC.C                                  *
167   *                                                                         *
168   * Edit History: 03/28/91 - Written by Robin T. Laird.                     *
169   *                                                                         *
170   \*************************************************************************/
171
172   #define MIN_RANDOM_TIME         2000    /* Wait no less than 2 seconds.     */
173   #define MAX_RANDOM_TIME         7000    /* Wait no more than 7 seconds.     */
174
175   void rtc_wait(duration)
176   unsigned long duration;
177   {
178     int t;
179     unsigned long t0;
180
181     rtc_error = AOK;              /* Assume function successful...            */
182
183     t0 = rtc_time();
184     if (duration == RTC_RANDOM_TIME)
185     {
186       /* Gen a random time >= MIN_RANDOM_TIME and <= MAX_RANDOM_TIME.         */
187       /* Make sure we don't wait any longer than MAX_RANDOM_TIME trying.      */
188
189       t = rand();
190       while (t < MIN_RANDOM_TIME || t > MAX_RANDOM_TIME)
191       {
192         t = rand();
193         if (rtc_time() > t0 + MAX_RANDOM_TIME) return;
194       }
195
196       /* Wait that amount of time.                                            */
197
198       while (rtc_time() < t0 + t);
199     }
200     else
201     {
202       /* Wait amount of time specified by parameter duration.                 */
203
204       while (rtc_time() < t0 + duration);
205     }
206   }
207
208
209   /*************************************************************************\
210   *                                rtc_time                                 *
211   *************************************************************************
212   *                                                                         *
213   * Function:     Routine to return time in milliseconds. The timer         *
214   *               period (as configured in the initialization routine)      *
215   *               is approximately 8.888...ms for an 11.0592 MHz CPU (or    *
216   *               6.666...ms for 14.7456 MHz CPU). Add whole portion of ms  *
217   *               value derived from tickcnt and fraction of timer period   *
218   *               derived from TIMER 0 to calculate actual time. This gives *
219   *               a time value that is accurate within +/- 1 ms.            *
```



```
220   *                                                                         *
221   * Input:        rtc_time();                                               *
222   *                                                                         *
223   * Output:       Returns absolute time value in milliseconds.              *
224   *                                                                         *
225   * Globals:      rtc_error : module RTC.C                                  *
226   *               tickcnt   : module RTC.C                                  *
227   *                                                                         *
228   * Edit History: 06/19/90 - Written by Robin T. Laird.                     *
229   *                                                                         *
230   \*************************************************************************/
231
232   /* Define divisor for calculation of time value (based on tickcnt).      */
233   /* E.g., OSCf/(12*timer roll-over*ms scale) = 11.0592e6/(12*8192*1000).   */
234   /* Play some games to avoid floating point math.                         */
235
236   #if defined(I80152)
237   #define MS_SCALE_FACTOR         20      /* Reduced fraction equivalent.     */
238   #define OSCF_SCALE_FACTOR       3       /* Reduced fraction equivalent.     */
239   #define MS_PER_COUNT            5       /* ms/count dividend.               */
240   #define COUNT_PER_MS            6144    /* count/ms divisor.                */
241   #else
242   #define MS_SCALE_FACTOR         80      /* Reduced fraction equivalent.     */
243   #define OSCF_SCALE_FACTOR       9       /* Reduced fraction equivalent.     */
244   #define MS_PER_COUNT            5       /* ms/count dividend.               */
245   #define COUNT_PER_MS            4608    /* count/ms divisor.                */
246   #endif
247
248   unsigned long rtc_time()
249   {
250     #if defined(IBMAT)
251     static struct timeb timebuf;
252     #else
253     static word count;
254     #endif
255
256     rtc_error = AOK;              /* Assume function successful...            */
257
258     #if !defined(IBMAT)
259
260     /* Calculate portion of timer period (in ms) based on current timer value.*/
261     /* Add this value to time based on tickcnt to give actual time.          */
262
263     count = ((((word)TH0<<5) | ((word)TL0&0x1F)) * MS_PER_COUNT) / COUNT_PER_MS;
264     return((unsigned long)((tickcnt*MS_SCALE_FACTOR)/OSCF_SCALE_FACTOR) + count);
265
266     #else
267
268     /* Get time and store in timebuf structure (this includes ms portion).   */
269     /* Return number of seconds * 1000 plus ms time for total ms time.       */
270
271     ftime(&timebuf);
272     return((unsigned long)timebuf.time*1000+timebuf.millitm);
273
274     #endif
275   }
```

```
/*****************************************************************************\
*                                   sio.h                                     *
*****************************************************************************
*                                                                           *
* CPCI:         IED90-MRA-COM-DEV-SIO-H-R0C3                                *
*                                                                           *
* Description:  Serial I/O device driver public declarations.               *
*               Contains the constant definitions and function prototypes   *
*               for the SIO.C module. These functions provide standard      *
*               access to the on-board serial I/O port.                     *
*                                                                           *
*               Module SIO exports the following variables/functions:       *
*                                                                           *
*               int sio_error;                                              *
*                                                                           *
*               sio_init();                                                 *
*               sio_putbyte();                                              *
*               sio_getbyte();                                              *
*               sio_byteavail();                                            *
*               sio_putstr();                                               *
*               sio_getstr();                                               *
*               sio_putnstr();                                              *
*                                                                           *
* Notes:        1) The SIO functions are part of the MRA standard           *
*                  hardware device driver (DEV) subsystem.                  *
*                                                                           *
* Edit History: 05/15/91 - Modified by Robin T. Laird.                      *
*                                                                           *
\*****************************************************************************/

/* Public Da a Structures:                                                  */

#ifndef SIO_MODULE_CODE
#define SIO_MODULE_CODE         6000

#define SIO_ERR_NOT_INIT        1+SIO_MODULE_CODE
#define SIO_ERR_BAUD_RATE       2+SIO_MODULE_CODE
#define SIO_ERR_PARITY          3+SIO_MODULE_CODE
#define SIO_ERR_WORD_LENGTH     4+SIO_MODULE_CODE
#define SIO_ERR_STOP_BITS       5+SIO_MODULE_CODE
#define SIO_ERR_STRING_LENGTH   6+SIO_MODULE_CODE

/* I80152 signifies baud rate settings for 14.7456 MHz CPU.                 */
/* I8031  signifies baud rate settings for 11.0592 MHz CPU.                 */
/* IBMAT signifies baud rate settings for IBM-PC/AT using 8250.             */
/* SBC8 signifies baud rate settings for Winsystems LPM-SBC8-4.             */

#if defined(I80152)
#define MAX_BAUD_RATE           0xFF    /* 38.4 KBPS (SMOD=0)               */
#define BR192                   0xFE
#define BR9600                  0xFC
#define BR4800                  0xF8
#define BR2400                  0xF0
#define BR1200                  0xE0
#define BR600                   0xC0
#define BR300                   0x80

#elif defined(I8031)
#define MAX_BAUD_RATE           0xFF    /* 57.6 KBPS (SMOD=1)               */
#define BR19200                 0xFD
#define BR9600                  0xFA
#define BR4800                  0xF4
#define BR2400                  0xE8
#define BR1200                  0xD0
#define BR600                   0xA0
#define BR300                   0x40

#elif defined(IBMAT)
#define BR38400                 3       /* Baud rate divisor settings.      */
#define BR19200                 6
#define BR9600                  12
#define BR4800                  24
#define BR2400                  48
```



```
#define BR1200                  96
#define BR600                   192
#define BR300                   384

#elif defined(SBC8)
#define BR38400                 0x7E    /* Baud rate CO SELECT=0 DIVISOR=4.  */
#define BR19200                 0x02
#define BR9600                  0x06
#define BR4800                  0x0E
#define BR2400                  0x16
#define BR1200                  0x22
#define BR600                   0x26
#define BR300                   0x2A
#endif

#if defined(I8031) || defined(I80152)
#define PEVEN                   0xC0    /* Parity settings (80535).         */
#define PODD                    0x80
#define PNONE                   0x00

#define WL5                     0xC0    /* Word length settings (80535).    */
#define WL6                     0x80
#define WL7                     0x40
#define WL8                     0x00

#define SB1                     0x00    /* Stop bit settings (80535).       */
#define SB2                     0x20

/* Local serial channel port identifier (8031 and 80152).                   */
/* Auxiliary serial channel port identifier (80535 only).                   */

#define LSC                     1
#define ASC                     2

#elif defined(IBMAT)
#define PEVEN                   0x18    /* Parity settings.                 */
#define PODD                    0x08
#define PNONE                   0x00

#define WL5                     0x00    /* Word length settings.            */
#define WL6                     0x01
#define WL7                     0x02
#define WL8                     0x03

#define SB1                     0x00    /* Stop bit settings.               */
#define SB2                     0x04

/* COM port base addresses.                                                 */

#define COM1                    0x3F0   /* 3F8-3FF.                         */
#define COM2                    0x2F0   /* 2F8-2FF.                         */
#define COM3                    0x3E0   /* 3E8-3EF.                         */
#define COM4                    0x2E0   /* 2E8-2EF.                         */

#elif defined(SBC8)
#define PEVEN                   0x00    /* Parity settings.                 */
#define PODD                    0x02
#define PNONE                   0x0C

#define WL5                     0x00    /* Word length settings.            */
#define WL6                     0x10
#define WL7                     0x20
#define WL8                     0x30

                                        /* Stop bit settings.               */
#define SB1                     0x00
#define SB2                     0x01

/* COM port base addresses.                                                 */

#define COM1                    0x90    /* LPM-SBC8 J1.                     */
#define COM2                    0x20    /* LPM-SIO4 J10.                    */
#define COM3                    0x24    /* LPM-SIO4 J8.                     */
#define COM4                    0x28    /* LPM-SIO4 J4.                     */
```

```
147   #define  OM5                    0x2C    /* LPM-SIO4 J3.                    */
148   #endif
149
150   /* External module global error variable.                                   */
151
152   extern int sio_error;
153
154   /* Public Functions:                                                        */
155
156   void sio_init(int port, int baud_rate, int parity, int word_len, int stop_bit);
157   void sio_putbyte(int port, byte c);
158   byte sio_getbyte(int port);
159   int  sio_byteavail(int port);
160   void sio_putstr(int port, byte *s);
161   int  sio_getstr(int port, byte *s);
162   void sio_putnstr(int port, int numbytes, byte *s);
163
164   #endif
```

```
1   /*****************************************************************************\
2   *                                  sio.c                                      *
3   *****************************************************************************
4   *                                                                            *
5   * CPCI:          IED90-MRA-COM-DEV-SIO-C-R0C3                                *
6   *                                                                            *
7   * Description:   Serial I/O device driver functions.                         *
8   *                Implements a standard set of serial I/O functions for       *
9   *                transmitting/receiving data over the local serial port.     *
10  *                The actual serial port used depends upon the implementation,*
11  *                but the functions and their prototypes must remain the same.*
12  *                Currently implemented for the Intel 8031, 80152, 80535, and *
13  *                IBM-AT (8250).                                              *
14  *                                                                            *
15  *                Module SIO exports the following variables/functions:       *
16  *                                                                            *
17  *                int sio_error;                                              *
18  *                                                                            *
19  *                sio_init();                                                 *
20  *                sio_putbyte();                                              *
21  *                sio_getbyte();                                              *
22  *                sio_byteavail();                                            *
23  *                sio_putstr();                                               *
24  *                sio_getstr();                                               *
25  *                sio_putnstr();                                              *
26  *                                                                            *
27  * Notes:         1) The SIO functions are part of the MRA standard           *
28  *                   hardware device driver (DEV) subsystem.                  *
29  *                2) See the Intel 8-Bit Embedded Controllers handbook        *
30  *                   for more information (No. 270645-002, pp. 7-11 - 7-21).  *
31  *                3) The serial I/O functions for the auxiliary serial channel*
32  *                   (ASC) of the CP-31/535 are interrupt driven, whereas all *
33  *                   other functions are based on polling.                    *
34  *                4) The CP-31/535 must be configured so that serial I/O      *
35  *                   interrupts of the 8256 (level L4 and L5) are "connected" *
36  *                   to the INT0 line of the 80535.                           *
37  *                                                                            *
38  * Edit History: 05/21/91 - Modified by Robin T. Laird.                       *
39  *                                                                            *
40  \*****************************************************************************/
41
42  #if defined(I80152) || defined(I8031)
43  #include <reg515.h>                     /* SFR definitions for 8031, etc.   */
44  #include <absacc.h>                     /* Absolute memory access defs.     */
45  #endif
46  #include <string.h>                     /* Standard string functions.       */
47  #include <sysdefs.h>                    /* System constants and types.      */
48  #include <sio.h>                        /* Serial I/O constants.            */
49
50  /* Public Variables:                                                        */
51
52  /* Global module error variable, sio_error.                                */
53  /* sio_error contains code of last error occurence.                         */
54  /* Should be set to AOK after each successful function call.                */
55  /* Variable can be examined by other software after each function call.     */
56
57  XDATA int sio_error = SIO_ERR_NOT_INIT; /* Global module error variable.    */
58
59  /* Communications port definitions for MPU (IBM-AT 8250).                   */
60
61  #if defined(IBMAT)
62
63  /* "8250" communications controller (CC) register address offsets.          */
64
65  #define CC_THR                  0x08    /* Transmit holding reg (write).    */
66  #define CC_RBR                  0x08    /* Receiver buffer reg (read).      */
67  #define CC_DLSB                 0x08    /* Divisor latch LSB.               */
68  #define CC_DMSB                 0x09    /* Divisor latch MSB.               */
69  #define CC_IER                  0x09    /* Interrupt enable register.       */
70  #define CC_IIR                  0x0A    /* Interrupt identification reg.    */
71  #define CC_LCR                  0x0B    /* Line control register.           */
72  #define CC_MCR                  0x0C    /* Modem control register.          */
73  #define CC_LSR                  0x0D    /* Line status register.            */
```



```
74  #define CC_MSR                  0x0E    /* Modem status register.           */
75
76  #define CC_DATA_READY           0x01    /* LSR data ready bit.              */
77  #define CC_RECEIVE_ERRS         0x1E    /* LSR error bits mask.             */
78  #define CC_TRANSMIT_HOLD_EMPTY  0x20    /* LSR xmtr holding register empty. */
79
80  #define CC_DLAB0                0x00    /* DLAB set low (0).                */
81  #define CC_DLAB1                0x80    /* DLAB set high (1).               */
82
83  #define CC_DISABLE_INT          0x00    /* Disable all interrupts.          */
84  #define CC_ENABLE_RCV           0x01    /* Enable receive interrupts.       */
85  #define CC_ENABLE_XMT           0x02    /* Enable transmit interrupts.      */
86  #define CC_ENABLE_STAT          0x04    /* Enable line status interrupts.   */
87  #define CC_MODEM_BUS_ENABLE     0x0B    /* DTR, RTS, and OUT2 active.       */
88
89  #elif defined(SBC8)
90
91  /* "8252" communications controller (CC) register address offsets.          */
92
93  #define CC_RBR                  0x00    /* Receiver buffer register.        */
94  #define CC_TBR                  0x00    /* Transmit buffer register.        */
95  #define CC_UCR                  0x01    /* UART control register.           */
96  #define CC_USR                  0x01    /* UART status register.            */
97  #define CC_MCR                  0x02    /* Modem control register.          */
98  #define CC_MSR                  0x03    /* Modem status register.           */
99  #define CC_BRSR                 0x03    /* Baud rate select register.       */
100
101 #define CC_MCR_RESET            0x0023  /* "Safe" MCR init value.           */
102 #define CC_RECV_ERR_MASK        0x000F  /* Receive error bit mask.          */
103 #define CC_TBRE_MASK            0x0040  /* Xfr-buff-ready bit mask.         */
104 #define CC_DR_MASK              0x0080  /* Data-ready bit mask.             */
105
106 /* Variables for saved USR, since with 8252 USR is reset after it's read.   */
107
108 static int old_usr = 0;
109 static int new_usr = 0;
110 static int byte_avail = FALSE;
111
112 /* "8256" MUART register definitions for CP-31/535 (80535).                 */
113
114 #else
115
116 /* 8256 MUART is addressed at absolute addresses F000H to FFFFH on 80535.   */
117
118 #define MUART_CMD_1     XBYTE[0xF000]   /* Command reg 1 (stop bit, parity).*/
119 #define MUART_CMD_2     XBYTE[0xF001]   /* Command reg 2 (baud rate).       */
120 #define MUART_CMD_3     XBYTE[0xF002]   /* Command reg 3 (RCV enable).      */
121
122 #define MUART_IER       XBYTE[0xF005]   /* Interrupt enable register.       */
123 #define MUART_IAR       XBYTE[0xF006]   /* Interrupt address register.      */
124 #define MUART_RBR       XBYTE[0xF007]   /* Data receive buffer.             */
125 #define MUART_TBR       XBYTE[0xF007]   /* Data transmit buffer.            */
126 #define MUART_MSR       XBYTE[0xF00F]   /* MUART status register.           */
127
128 #define MUART_RCV_ENABLE        0xC0    /* RCV enable, SET bit hi.          */
129 #define MUART_NESTED_INTS       0x90    /* Nested ints enabled, SET bit hi. */
130 #define MUART_EOI               0x88    /* End-of-interrupt, SET bit hi.    */
131
132 #define MUART_RECV_ERR_MASK     0x07    /* RCV errors in LSB of MSR.        */
133 #define MUART_TBE_MASK          0x20    /* Transmit buffer empty in MSR.    */
134 #define MUART_RBF_MASK          0x40    /* Receive buffer full in MSR.      */
135
136 #define MUART_RCV_INT_ENABLE    0x10    /* Level 4 interrupt enable.        */
137 #define MUART_XMT_INT_ENABLE    0x20    /* Level 5 interrupt enable.        */
138
139 #define MUART_RCV_INT_MASK      0x10    /* Level 4 interrupt identifier.    */
140 #define MUART_XMT_INT_MASK      0x14    /* Level 5 interrupt identifier.    */
141
142 /* Baud rate settings (which are different than those for 8031/80152).      */
143
144 #define _BR19200                0x03    /* Internal baud rate generator.    */
145 #define _BR9600                 0x04
146 #define _BR4800                 0x05
```

```
147  #define _BR2400                 0x06
148  #define _BR1200                 0x07
149  #define _BR600                  0x08
150  #define _BR300                  0x09
151
152  /* Type definition for I/O data queue.                                    */
153
154  #define MAX_BUFFER_SIZE         128
155
156  typedef struct { byte item[MAX_BUFFER_SIZE];
157                   word front;
158                   word rear;
159                   byte empty;
160                   byte full;
161                 } buffer;
162
163  /* Input (receive) and output (transmit) global data queues.             */
164
165  static XDATA buffer xmt_buffer;
166  static XDATA buffer rcv_buffer;
167
168  #endif
169
170
171  /*************************************************************************\
172  *                                buf_inc                                  *
173  *************************************************************************
174  *                                                                         *
175  * Function:     Macro that increments either the front or rear index      *
176  *               of a circular buffer of max size MAX_BUFFER_SIZE.         *
177  *               Should be used with care, i.e., not nested within a       *
178  *               compound statement like if ... then ... else.             *
179  *                                                                         *
180  * Input:        buf_inc(                                                  *
181  *                 word x; index to increment.                             *
182  *               );                                                        *
183  *                                                                         *
184  * Output:       Nothing.                                                  *
185  *                                                                         *
186  * Globals:      None.                                                     *
187  *                                                                         *
188  * Edit History: 12/04/90 - Written by Robin T. Laird.                     *
189  *                                                                         *
190  \*************************************************************************/
191
192  #define buf_inc(x) (x = ((x == MAX_BUFFER_SIZE-1) ? 0 : x+1))
193
194
195  /*************************************************************************\
196  *                                sio_init                                 *
197  *************************************************************************
198  *                                                                         *
199  * Function:     Serial port initialization function. Sets up the          *
200  *               indicated serial port to operate according to the         *
201  *               specified parameters (e.g., baud rate, etc.).             *
202  *                                                                         *
203  *               For the 8031, the following settings are forced:          *
204  *                                                                         *
205  *               parity      : PNONE (no parity)                           *
206  *               word_len    : WL8   (8 data bits)                         *
207  *               stop_bits   : SB1   (1 stop bit)                          *
208  *                                                                         *
209  * Input:        sio_init(                                                 *
210  *                 int port;      serial port number (identifier).         *
211  *                 int baud_rate; serial port baud rate.                   *
212  *                 int parity;    serial port parity.                      *
213  *                 int word_len;  serial port word length.                 *
214  *                 int stop_bit;  serial port stop bits.                   *
215  *               );                                                        *
216  *                                                                         *
217  * Output:       Nothing.                                                  *
218  *                                                                         *
219  * Globals:      sio_error : module SIO.C                                  *
```



```
220  *               8031 regs : module REG515.H                               *
221  *                                                                         *
222  * Edit History: 10/10/86 - Written by Robin T. Laird.                     *
223  *                                                                         *
224  \*************************************************************************/
225
226  /* 8031 register values for initializing the local serial channel (LSC).  */
227  /* If using higher clock speed, SMOD is set to 0, otherwise set to 1.     */
228  /* SMOD is the MSB of the PCON Power Control register on the 8031.        */
229
230  #if defined(I80152) || defined(I8031)
231
232  #define SERIAL_PORT_MODE_1      0x50            /* SCON = 0bx1x1xxxx    */
233  #define SERIAL_PORT_MASK        0x5F            /* SCON = 0b0101xxxx    */
234  #define TIMER_1_MODE_2          0x20            /* TMOD = 0bxx1xxxxx    */
235  #define TIMER_1_MASK            0x2F            /* TMOD = 0b0010xxxx    */
236
237  #if defined(I80152)
238  #define DOUBLE_BAUD_RATE        0x00            /* PCON = 0b0xxxxxxx SMOD=0 */
239  #else
240  #define DOUBLE_BAUD_RATE        0x80            /* PCON = 0b1xxxxxxx SMOD=1 */
241  #endif
242
243  /* IBM-AT (8250) clear receive buffer attempt count.                      */
244
245  #elif defined(IBMAT)
246
247  #define MAX_RFIFO_READS         10              /* Clear this many bytes. */
248
249  #endif
250
251  void sio_init(port, baud_rate, parity, word_len, stop_bit)
252  int port;
253  int baud_rate;
254  int parity;
255  int word_len;
256  int stop_bit;
257  {
258    int i;
259
260    sio_error = AOK;              /* Assume function successful...        */
261
262    /* Check settings for validity.                                         */
263
264    switch(baud_rate)
265    {
266      #if defined(IBMAT) || defined(SBC8)
267      case BR38400:
268      #elif defined(I80152) || defined(I8031)
269      case MAX_BAUD_RATE:
270      #endif
271      case BR19200:
272      case BR9600:
273      case BR4800:
274      case BR2400:
275      case BR1200:
276      case BR600:
277      case BR300:
278        break;
279      default:
280        sio_error = SIO_ERR_BAUD_RATE;
281        return;
282    }
283
284    switch(parity)
285    {
286      case PEVEN:
287      case PODD:
288      case PNONE:
289        break;
290      default:
291        sio_error = SIO_ERR_PARITY;
292        return;
```

```
293     }
294
295     switch(word_len)
296     {
297       case WL5:
298       case WL6:
299       case WL7:
300       case WL8:
301         break;
302       default:
303         sio_error = SIO_ERR_WORD_LENGTH;
304         return;
305     }
306
307     switch(stop_bit)
308     {
309       case SB1:
310       case SB2:
311         break;
312       default:
313         sio_error = SIO_ERR_STOP_BITS;
314         return;
315     }
316
317     #if defined(I80152) || defined(I8031)
318
319     /* Local and auxiliary serial channels are initialized differently.      */
320
321     if (port == ASC)
322     {
323       /* Determine 8256 baud rate from parameter baud rate.                  */
324
325       switch(baud_rate)
326       {
327         case MAX_BAUD_RATE:
328         case BR19200:
329           baud_rate = _BR19200;
330           break;
331         case BR9600:
332           baud_rate = _BR9600;
333           break;
334         case BR4800:
335           baud_rate = _BR4800;
336           break;
337         case BR2400:
338           baud_rate = _BR2400;
339           break;
340         case BR1200:
341           baud_rate = _BR1200;
342           break;
343         case BR600:
344           baud_rate = _BR600;
345           break;
346         case BR300:
347           baud_rate = _BR300;
348           break;
349         default:
350           baud_rate = _BR19200;
351           break;
352       }
353
354
355       /* Init COMMAND REGISTER 1: stop bits, character length.               */
356       /* Init COMMAND REGISTER 2: baud rate, parity.                         */
357       /* Init COMMAND REGISTER 3: receiver enable.                           */
358       /* Clear the receive buffer.                                           */
359
360       MUART_CMD_1 = word_len | stop_bit;
361       MUART_CMD_2 = parity   | baud_rate;
362       MUART_CMD_3 = MUART_RCV_ENABLE;
363       while ((MUART_MSR & MUART_RBF_MASK) == MUART_RBF_MASK) i = MUART_RBR;
364
365       #if !defined(MUART_POLLED)
```



```
366
367       /* Init I/O data queue structures.                                     */
368
369       xmt_buffer.front = xmt_buffer.rear = 0;
370       xmt_buffer.empty = TRUE;
371       xmt_buffer.full  = FALSE;
372
373       rcv_buffer.front = rcv_buffer.rear = 0;
374       rcv_buffer.empty = TRUE;
375       rcv_buffer.full  = FALSE;
376
377       /* Indicate low-level triggered interrupt from 8256.                   */
378       /* Enable external interrupt 0 (EX0) on CP-31/535 80535.               */
379
380       IT0 = 0;
381       EX0 = 1;
382
383       /* Enable receive and transmit interrupts on the CP-31/535 8256.       */
384       /* Assumes some other function enables processor interrupts.           */
385
386       MUART_CMD_3 = MUART_NESTED_INTS;
387       MUART_IER  |= MUART_RCV_INT_ENABLE;
388       MUART_IER  |= MUART_XMT_INT_ENABLE;
389
390       #endif
391     }
392     else
393     {
394       /* Set registers according to chosen options.                          */
395       /* TIMER 1 is used in MODE 2 for variable baud rates.                  */
396       /* TH1 sets TIMER 1 re-load value.                                     */
397       /* SCON controls the serial port modes settings.                       */
398       /* TMOD controls the timer mode settings.                              */
399       /* TCON controls the timer itself (turns it on/off).                   */
400       /* SMOD controls baud rate doubling depending upon CPU speed.          */
401
402       PCON |= DOUBLE_BAUD_RATE;   /* Set Power Control Mode register.        */
403       TH1   = baud_rate;          /* Set TIMER 1 re-load value for baud.     */
404       TMOD |= TIMER_1_MODE_2;     /* Set Timer Mode Control register.        */
405       TMOD &= TIMER_1_MASK;
406       SCON |= SERIAL_PORT_MODE_1; /* Set Serial Port Control register.       */
407       SCON &= SERIAL_PORT_MASK;
408       TR1   = 1;                  /* Set TIMER 1 run bit.                    */
409     }
410
411     #elif defined(IBMAT)
412
413     /* Initialize 8250, see "Technical Reference Personal Computer AT".      */
414     /* Set baud rate (divisor latches).                                      */
415     /* DELAB must be 1 to write to baud rate divisor latches.                */
416     /* Parameter baud_rate is ignored.                                       */
417
418     outp(port+CC_LCR, CC_DLAB1);
419     outp(port+CC_DMSB, (baud_rate >> 8) & 0xFF);
420     outp(port+CC_DLSB, baud_rate & 0xFF);
421
422     /* Set serial port operational parameters.                               */
423     /* DELAB must be 0 to write/read LCR, IER, and data registers.           */
424
425     outp(port+CC_LCR, CC_DLAB0 | parity | word_len | stop_bit);
426
427     /* Disable all 8250 interrupts on this port.                             */
428
429     outp(port+CC_IER, CC_DISABLE_INT);
430
431     /* Clear line status and modem status registers.                         */
432
433     inp(port+CC_LSR);
434     inp(port+CC_MSR);
435
436     /* Clear chars from receive register.                                    */
437     /* Line status register (bit 0) indicates if data ready.                 */
438
```

```
439     for (i = 0; i < MAX_RFIFO_READS; i++)
440     {
441      if (inp(port+CC_LSR) & CC_DATA_READY)
442        inp(port+CC_RBR);
443      else
444        break;
445     }
446     if (i > MAX_RFIFO_READS || inp(port+CC_LSR) & CC_DATA_READY)
447     {
448       sio_error = SIO_ERR_NOT_INIT;
449       return;
450     }
451
452     #elif defined(SBC8)
453
454     /* Set 8252 baud rate divisor (and CO SELECT).                          */
455     /* Set 8252 UART control register.                                      */
456     /* Re-set 8252 modem control register.                                  */
457
458     outp(port+CC_BRSR, baud_rate);
459     outp(port+CC_UCR,  parity | word_len | stop_bit);
460     outp(port+CC_MCR,  CC_MCR_RESET);
461     #endif
462   }
463
464
465   /*************************************************************************\
466   *                              sio_putbyte                              *
467   *************************************************************************
468   *                                                                       *
469   * Function:     Output a byte to the specified serial port.             *
470   *               Byte is output to the transmit buffer of the port.      *
471   *                                                                       *
472   * Input:        sio_putbyte(                                            *
473   *                 int port; serial port number (identifier).            *
474   *                 byte c;   character (value) to output.                *
475   *               );                                                      *
476   *                                                                       *
477   * Output:       Nothing.                                                *
478   *                                                                       *
479   * Globals:      sio_error : module SIO.C                                *
480   *               803I regs : module REG515.H                             *
481   *                                                                       *
482   * Edit History: 10/10/86 - Written by Robin T. Laird.                   *
483   *                                                                       *
484   \*************************************************************************/
485
486   void sio_putbyte(port, c)
487   int  port;
488   byte c;
489   {
490     /* Output value to serial port.                                       */
491     /* Wait for character to be sent (transmit buffer empty).             */
492
493     sio_error = AOK;              /* Assume function successful...        */
494
495     #if defined(180152) || defined(18031)
496
497     /* Local and auxiliary serial channels are managed differently.       */
498
499     if (port == ASC)
500     {
501       #if defined(MUART_POLLED)
502
503       while((MUART_MSR & MUART_TBE_MASK) != MUART_TBE_MASK);
504       MUART_TBR = c;
505
506       #else
507
508       /* If buffer full, then wait until it's not.                        */
509
510       while (xmt_buffer.full);
511
```



```
512       /* Place byte in output queue.                                      */
513       /* Adjust rear index to account for added byte.                     */
514       /* See if buffer is full.                                           */
515       /* If transmit buffer is empty, move byte to transmitter (generate INT).*/
516       /* Buffer can't be empty since we just added a byte.                */
517
518       xmt_buffer.item[xmt_buffer.rear] = c;
519       buf_inc(xmt_buffer.rear);
520       if (xmt_buffer.front == xmt_buffer.rear) xmt_buffer.full = TRUE;
521       if (xmt_buffer.empty)
522       {
523         xmt_buffer.empty = FALSE;
524         MUART_TBR = c;
525       }
526       else
527         xmt_buffer.empty = FALSE;
528
529       #endif
530     }
531     else
532     {
533       SBUF = c;                   /* Put byte into serial buffer.         */
534       while(TI != 1);             /* Wait for transmit interrupt flag to set. */
535       TI = 0;                     /* Clear interrupt flag for next xmit.  */
536     }
537
538     #elif defined(IBMAT)
539
540     outo(port+CC_THR, c);
541     while((inp(port+CC_LSR) & CC_TRANSMIT_HOLD_EMPTY) == 0);
542
543     #elif defined(SBC8)
544
545     /* Output value to serial port.                                       */
546     /* Wait for character to be sent (transmit buffer empty).             */
547     /* Need to save status of USR if byte avail.                          */
548
549     outp(port+CC_TBR, c);
550     do {
551       new_usr = inp(port+CC_USR);
552       if ((new_usr & CC_DR_MASK) == CC_DR_MASK) old_usr = new_usr;
553     } while((inp(port+CC_USR) & CC_TBRE_MASK) != CC_TBRE_MASK);
554
555     #endif
556   }
557
558
559   /*************************************************************************\
560   *                              sio_getbyte                              *
561   *************************************************************************
562   *                                                                       *
563   * Function:     Input character from specified serial port.             *
564   *               Character is input from data register.                  *
565   *               Waits forever for input.                                *
566   *                                                                       *
567   * Input:        sio_getbyte(                                            *
568   *                 int port; serial port number (identifier).            *
569   *               );                                                      *
570   *                                                                       *
571   * Output:       Returns character read from serial port.                *
572   *                                                                       *
573   * Globals:      sio_error : module SIO.C                                *
574   *               803I regs : module REG515.H                             *
575   *                                                                       *
576   * Edit History: 10/10/86 - Written by Robin T. Laird.                   *
577   *                                                                       *
578   \*************************************************************************/
579
580   byte sio_getbyte(port)
581   int port;
582   {
583     byte c;
584
```

```
585     /* Wait until there is a character available to input.                  */
586     /* Get character from port.                                             */
587     /* Return input character (as a byte).                                  */
588
589     sio_error = AOK;              /* Assume function successful...          */
590
591     #if defined(I80152) || defined(I8031)
592
593     /* Local and auxiliary serial channels are managed differently.         */
594
595     if (port == ASC)
596     {
597       #if defined(MUART_POLLED)
598
599       while ((MUART_MSR & MUART_RBF_MASK) != MUART_RBF_MASK);
600       return(MUART_RBR);
601
602       #else
603
604       /* If the buffer is empty, then wait for a char to be received.       */
605
606       while(rcv_buffer.empty);
607
608       /* Remove byte from buffer.                                           */
609       /* Adjust front index to account for byte removed.                    */
610       /* Buffer can't be full since we just removed a byte.                 */
611       /* See if the buffer is empty.                                        */
612
613       c = rcv_buffer.item[rcv_buffer.front];
614       buf_inc(rcv_buffer.front);
615       rcv_buffer.full = FALSE;
616       if (rcv_buffer.front == rcv_buffer.rear) rcv_buffer.empty = TRUE;
617       return(c);
618
619       #endif
620     }
621     else
622     {
623       while (RI != 1);            /* Wait for receive interrupt flag to set.*/
624       RI = 0;                     /* Clear interrupt flag for next recv.    */
625       return(SBUF);               /* Return byte in serial buffer.          */
626     }
627
628     #elif defined(IBMAT)
629
630     while ((inp(port+CC_LSR) & CC_DATA_READY) == 0);
631     return(inp(port+CC_RBR));
632
633     #elif defined(SBC8)
634
635     /* Indicate that byte no longer available.                              */
636     /* Get current value of USR.                                            */
637     /* If USR value is 0, then use saved USR value.                         */
638     /* Otherwise (if USR != 0), wait until a byte is avail, and return it.  */
639
640     byte_avail = FALSE;
641     new_usr = inp(port+CC_USR);
642     if (new_usr == 0)
643     {
644       if ((old_usr & CC_DR_MASK) == CC_DR_MASK)
645       {
646         old_usr = 0;
647         return(inp(port+CC_RBR));
648       }
649       else
650       {
651         while ((inp(port+CC_USR) & CC_DR_MASK) != CC_DR_MASK);
652         return(inp(port+CC_RBR));
653       }
654     }
655     else if ((new_usr & CC_DR_MASK) == CC_DR_MASK)
656       return(inp(port+CC_RBR));
657
```



```
658     while ((inp(port+CC_USR) & CC_DR_MASK) != CC_DR_MASK);
659     return(inp(port+CC_RBR));
660
661     #endif
662   }
663
664
665   /*****************************************************************************\
666   *                                sio_byteavail                                *
667   *****************************************************************************
668   *                                                                           *
669   * Function:     Determines whether or not a byte is available at the serial *
670   *               port. Does not remove the byte. Will return TRUE until the  *
671   *               byte is removed and no new byte is received.                *
672   *                                                                           *
673   * Input:        sio_byteavail(                                              *
674   *                 int port; serial port number (identifier).                *
675   *               );                                                          *
676   *                                                                           *
677   * Output:       Returns TRUE if byte availabe, otherwise returns FALSE.     *
678   *                                                                           *
679   * Globals:      sio_error : module SIO.C                                    *
680   *               8031 regs : module REG515.H                                 *
681   *                                                                           *
682   * Edit History: 10/10/86 - Written by Robin T. Laird.                       *
683   *                                                                           *
684   \*****************************************************************************/
685
686   int sio_byteavail(port)
687   int port;
688   {
689     sio_error = AOK;              /* Assume function successful...          */
690
691     #if defined(I80152) || defined(I8031)
692
693     /* Local and auxiliary serial channels are managed differently.         */
694
695     if (port == ASC)
696     {
697       #if defined(MUART_POLLED)
698
699       /* Check MUART status register receive buffer full flag.              */
700
701       return(MUART_MSR & MUART_RBF_MASK ? TRUE : FALSE);
702
703       #else
704
705       /* Check receive buffer empty flag.                                   */
706
707       return(!rcv_buffer.empty);
708
709       #endif
710     }
711     else
712     {
713       /* Check RI for byte availability.                                    */
714
715       return(RI == 1 ? TRUE : FALSE);
716     }
717
718     #elif defined(IBMAT)
719
720     /* Check data ready for byte. Data ready reset by read of rcv buffer.   */
721
722     return((inp(port+CC_LSR) & CC_DATA_READY) == 1 ? TRUE : FALSE);
723
724     #elif defined(SBC8)
725
726     /* Since USR (input status reg) is reset on read we have to play games. */
727     /* Check byte avail flag to see if set from previous call.              */
728     /* If flag not set, get current value of USR.                           */
729     /* Check old value of USR to see if byte was available.                 */
730
```

```
731     if (byte_avail)
732       return(TRUE);
733     else
734     {
735       new_usr = inp(port+CC_USR);
736       if ((old_usr & CC_DR_MASK) == CC_DR_MASK)
737       {
738         byte_avail = TRUE;
739       }
740       else
741       {
742         old_usr = new_usr;
743         byte_avail = (new_usr & CC_DR_MASK) ? TRUE : FALSE;
744       }
745       return(byte_avail);
746     }
747
748   #endif
749   }
750
751
752   /*****************************************************************************\
753   *                                 sio_putstr                                  *
754   ******************************************************************************
755   *                                                                             *
756   * Function:     Outputs a NULL terminated string to the serial port.          *
757   *                                                                             *
758   * Input:        sio_putstr(                                                   *
759   *                 byte *s;  pointer to string to output.                      *
760   *                 int port; serial port number (identifier).                  *
761   *               );                                                            *
762   *                                                                             *
763   * Output:       Nothing.                                                      *
764   *                                                                             *
765   * Globals:      sio_error : module SIO.C                                      *
766   *                                                                             *
767   * Edit History: 10/10/86 - Written by Robin T. Laird.                         *
768   *                                                                             *
769   \*****************************************************************************/
770
771   void sio_putstr(port, s)
772   int  port;
773   byte *s;
774   {
775     byte *stemp;
776     word room_in_queue;
777
778     /* Output string, char by char (as done above) until NULL char.          */
779
780     sio_error = AOK;              /* Assume function successful...           */
781
782     #if defined(I8031)
783
784     if (port == ASC)
785     {
786       /* Wait until there's enough room for string to be inserted into queue. */
787
788       do {
789         if (xmt_buffer.front <= xmt_buffer.rear)
790           room_in_queue = MAX_BUFFER_SIZE - xmt_buffer.rear + xmt_buffer.front;
791         else
792           room_in_queue = xmt_buffer.front - xmt_buffer.rear - 1;
793       } while (room_in_queue < strlen((char*)s));
794
795       /* Copy output string to transmit queue (save original string ptr).    */
796
797       stemp = s;
798       while(*s)
799       {
800         xmt_buffer.item[xmt_buffer.rear] = *s++;
801         buf_inc(xmt_buffer.rear);
802       }
803
```



```
804       /* If transmit buffer was empty, move byte to transmitter (gen INT).  */
805       /* Buffer can't be empty since we just added a byte.                   */
806
807       if (xmt_buffer.empty)
808       {
809         xmt_buffer.empty = FALSE;
810         MUART_TBR = *stemp;
811       }
812       else
813         xmt_buffer.empty = FALSE;
814       return;
815     }
816
817     #endif
818
819     while(*s) sio_putbyte(port, *s++);
820   }
821
822
823   /*****************************************************************************\
824   *                                 sio_getstr                                  *
825   ******************************************************************************
826   *                                                                             *
827   * Function:     Inputs a NULL terminated string from the serial port.         *
828   *                                                                             *
829   * Input:        sio_getstr(                                                   *
830   *                 int port; serial port number (identifier).                  *
831   *                 char *s;  pointer to string where bytes will be placed.     *
832   *               );                                                            *
833   *                                                                             *
834   * Output:       Length of input string as an integer.                         *
835   *                                                                             *
836   * Globals:      sio_error : module SIO.C                                      *
837   *                                                                             *
838   * Edit History: 10/10/86 - Written by Robin T. Laird.                         *
839   *                                                                             *
840   \*****************************************************************************/
841
842   int sio_getstr(port, s)
843   int  port;
844   byte *s;
845   {
846   int i;
847
848     /* Input chars (as done above) until we hit a NULL char.                 */
849     /* Stuff chars into s, using s as string pointer.                        */
850     /* Keep track of string length in i, return i.                           */
851
852     sio_error = AOK;              /* Assume function successful...           */
853
854     i = 0;
855     do {
856       *s = sio_getbyte(port);
857       i++;
858     } while(*s++);
859     return(i);
860   }
861
862
863   /*****************************************************************************\
864   *                                 sio_putnstr                                 *
865   ******************************************************************************
866   *                                                                             *
867   * Function:     Outputs a (byte) string of numbytes in length.                *
868   *               Sets sio_error to SIO_ERR_STRING_LENGTH if the number of      *
869   *               bytes to send is less than or equal to 0.                     *
870   *                                                                             *
871   * Input:        sio_putnstr(                                                  *
872   *                 int port;     serial port number (identifier).            *
873   *                 int numbytes; number of bytes to output (numbytes > 0).   *
874   *                 byte *s;      pointer to (byte) string to output.         *
875   *                                                                             *
876   * Output:       Nothing.                                                      *
```

```
877   *                                                                          *
878   * Globals:      sio_error : module SIO.C                                   *
879   *                                                                          *
880   * Edit History: 06/19/90 - Written by Robin T. Laird.                      *
881   *                                                                          *
882   \**************************************************************************/
883
884   void sio_putnstr(port, numbytes, s)
885   int  port;
886   int  numbytes;
887   byte *s;
888   {
889     byte *stemp;
890     word room_in_queue;
891
892     /* Check the length of the string and indicate if invalid.              */
893
894     if (numbytes <= 0)
895     {
896       sio_error = SIO_ERR_STRING_LENGTH;
897     }
898     else
899     {
900       sio_error = AOK;
901
902       #if defined(I8031)
903
904       if (port == ASC)
905       {
906         /* Wait until there's enough room for string to be inserted into que. */
907
908         do {
909           if (xmt_buffer.front <= xmt_buffer.rear)
910             room_in_queue = MAX_BUFFER_SIZE - xmt_buffer.rear + xmt_buffer.front;
911           else
912             room_in_queue = xmt_buffer.front - xmt_buffer.rear - 1;
913         } while (room_in_queue < numbytes);
914
915         /* Copy output string to transmit queue (save original string ptr).   */
916
917         stemp = s;
918         while(numbytes--)
919         {
920           xmt_buffer.item[xmt_buffer.rear] = *s++;
921           buf_inc(xmt_buffer.rear);
922         }
923
924         /* If transmit buffer was empty, move byte to transmitter (gen INT).  */
925         /* Buffer can't be empty since we just added a byte.                   */
926
927         if (xmt_buffer.empty)
928         {
929           xmt_buffer.empty = FALSE;
930           MUART_TBR = *stemp;
931         }
932         else
933           xmt_buffer.empty = FALSE;
934         return;
935       }
936
937       #endif
938
939       while (numbytes--) sio_putbyte(port, *s++);
940     }
941   }
942
943
944   /**************************************************************************\
945   *                              sio_8256_int                                *
946   ****************************************************************************
947   *                                                                          *
948   * Function:     Handles transmit/receive serial interrupts from the 8256.  *
949   *               Assumes the use of the CP-31/535, and that the serial rcv  *
```



```
950   *               and xmt interrupts of the 8256 peripheral IC (level L4/L5) *
951   *               are connected to the INT0 interrupt input of the 80535.    *
952   *               (This requires jumpering the bottom two pins of W3 on the  *
953   *               CP-31/535 which connects 8256 INT to 80535 EXT INT0 input).*
954   *                                                                          *
955   *               Serial INPUT interrupts (for chars being received) move    *
956   *               incoming data to a global queue where it can be removed by *
957   *               the other serial I/O routines (e.g., sio_getbyte()).       *
958   *                                                                          *
959   *               Serial OUTPUT interrupts (for chars being transmitted) move*
960   *               outgoing data from a global queue that is filled by the    *
961   *               other serial I/O routines (e.g., sio_putbyte()).           *
962   *                                                                          *
963   *               Input (RCV) interrupts are distinguished from output (XMT) *
964   *               interrupts by reading the 8256 interrupt address register  *
965   *               which holds a value equal to the interrupt level * 4.      *
966   *                                                                          *
967   * Input:        sio_8256_int();                                            *
968   *                                                                          *
969   * Output:       Nothing.                                                   *
970   *                                                                          *
971   * Globals:      sio_error   : module SIO.C                                 *
972   *               sio_in_que  : module SIO.C                                 *
973   *               sio_out_que : module SIO.C                                 *
974   *                                                                          *
975   * Edit History: 05/21/91 - Written by Robin T. Laird.                      *
976   *                                                                          *
977   \**************************************************************************/
978
979   #if defined(I8031)
980
981   static void sio_8256_int() interrupt 0 using 2
982   {
983     static byte iar;
984
985     /* Get interrupt address register.                                      */
986
987     iar = MUART_IAR;
988
989     /* Check for receive (input) interrupt.                                 */
990     /* Reading IAR clears int and indicates INT ACK to int controller.      */
991
992     if (iar == MUART_RCV_INT_MASK)
993     {
994       /* Make sure receive buffer is not full.                              */
995       /* Move byte from input register to receive buffer.                   */
996       /* Adjust buffer rear index.                                          */
997       /* See if buffer is full.                                             */
998       /* Mark buffer as not empty since we just received something.         */
999
1000      if (!rcv_buffer.full)
1001      {
1002        rcv_buffer.item[rcv_buffer.rear] = MUART_RBR;
1003        buf_inc(rcv_buffer.rear);
1004        if (rcv_buffer.front == rcv_buffer.rear) rcv_buffer.full = TRUE;
1005        rcv_buffer.empty = FALSE;
1006      }
1007    }
1008
1009    /* Check for transmit (output) interrupt.                               */
1010
1011    else if (iar == MUART_XMT_INT_MASK)
1012    {
1013      /* Mark buffer as not full since we just removed an item.             */
1014      /* Byte has been transmitted so adjust front index accordingly.       */
1015      /* Check to see if buffer is empty. Set empty flag if so.             */
1016      /* If not empty, move byte from front of buffer to transmit register. */
1017
1018      if (!xmt_buffer.empty)
1019      {
1020        xmt_buffer.full = FALSE;
1021        buf_inc(xmt_buffer.front);
1022        if (xmt_buffer.front == xmt_buffer.rear)
```

```
1023            xmt_buffer.empty = TRUE;
1024          else
1025            MUART_TBR = xmt_buffer.item[xmt_buffer.front];
1026        }
1027      }
1028
1029      /* Issue End-of-Interrupt (EOI).                                    */
1030
1031      MUART_CMD_3 = MUART_EOI;
1032    }
1033
1034    #endif
```

```
1     /*****************************************************************************\
2     *                                  DEBUG.H                                    *
3     *****************************************************************************
4     *                                                                            *
5     * CPCI:          IED90-MRA-COM-HDR-DEBUG-H-R0C0                              *
6     *                                                                            *
7     * Description:   Global debug definitions/declarations.                      *
8     *                Includes headers required to output debug messages to the   *
9     *                local serial port (or standard output device) of the target *
10    *                system.                                                     *
11    *                                                                            *
12    * Notes:         1) All modules should include this file for debugging.      *
13    *                2) XDATA modifier not required for Microsoft C (MSC).       *
14    *                                                                            *
15    * Edit History: 03/22/91 - Written by Robin T. Laird.                        *
16    *                                                                            *
17    \*****************************************************************************/
18
19    #if !defined(IBMAT)
20    #include <reg51.h>
21    #endif
22    #include <stdio.h>
23    #include <sio.h>
24
25    static XDATA char spf[80];              /* Buffer for sprintf() calls.      */
```

```
1     /*****************************************************************************\
2     *                                SYSDEFS.H                                    *
3     *****************************************************************************
4     *                                                                           *
5     * CPCI:          IED90-MRA-COM-HDR-SYSDEFS-H-R0C0                           *
6     *                                                                           *
7     * Description:   Global system definitions/declarations.                    *
8     *                Defines absolute and common values used by all modules.    *
9     *                                                                           *
10    * Notes:         1) All modules should include this file.                   *
11    *                2) XDATA modifier not required for Microsoft C (MSC).      *
12    *                                                                           *
13    * Edit History: 07/07/90 - Written by Robin T. Laird.                       *
14    *                                                                           *
15    \*****************************************************************************/
16
17    #ifndef SYS_MODULE_CODE
18    #define SYS_MODULE_CODE          0
19
20    #define AOK                      1
21
22    #define TRUE                     1
23    #define FALSE                    0
24
25    #define YES                      1
26    #define NO                       0
27
28    #if !defined(NULL)
29    #define NULL                     0
30    #endif
31
32    #if !defined(NULLPTR)
33    #define NULLPTR                  ((char*)0)
34    #endif
35
36    #define SYS_MAX_PACKET_SIZE      256
37
38    #if defined(MSDOS)
39    #define XDATA
40    #else
41    #define XDATA                    xdata
42    #endif
43
44    typedef unsigned char byte;
45    typedef unsigned int word;
46
47    #endif
```

```
1   #*****************************************************************************#
2   #                                MAKEFILE                                     #
3   #*****************************************************************************#
4   #                                                                             #
5   # CPCI:          IED90-MRA-COM-LCS-MAKEFILE-TXT-R0C0                          #
6   #                                                                             #
7   # Description:   Makefile for the Modular Robotic Architecture (MRA).        #
8   #                Makes the common local communications subsystem.             #
9   #                                                                             #
10  #                Targets are available for the following systems/subsystems:  #
11  #                                                                             #
12  #                lcs    - COM Local Communications Subsystem                  #
13  #                lib    - Add modules to MRA library                          #
14  #                print  - Print COM local communications files                #
15  #                                                                             #
16  # Notes:         1) The dependency and production rules are included here.    #
17  #                2) See also \mra\makefile.                                   #
18  #                                                                             #
19  # Edit History: 0?/??/91 - Written by Robin T. Laird.                         #
20  #                                                                             #
21  #*****************************************************************************#
22
23
24
25  ##################################### RULES #####################################
26
27  .SUFFIXES : .hex .exe .obj .c .a51
28
29  # Control settings for Franklin 8031 development
30
31  CC          =c51
32  AS          =a51
33  LINK        =l51
34  OTOH        =ohs51
35  CFLAGS      =cd la db sb
36  ASFLAGS     =
37  LFLAGS      =
38  OFLAGS      =
39  STARTUP     =\c51\crom.obj
40  CODESEG     =00000h
41  XDATASEG    =00000h
42
43  # Control settings for Microsoft MS-DOS development
44
45  MSC         =cl
46  MSAS        =masm
47  MSLINK      =link
48  MSCFLAGS    =/AL /c /Oi /Zi /Od
49  MSASFLAGS   =
50  MSLNKFLGS   =/co
51  LOADLIBES   =
52
53  .c.obj :
54          $(CC) $< $(CFLAGS)
55
56  .a51.obj :
57          $(AS) $< $(ASFLAGS)
58
59  .obj.exe :
60          $(LINK) $(STARTUP),$< TO $@ code ($(CODESEG)) xdata ($(XDATASEG)) ixref
61
62  .exe.hex :
63          $(OTOH) $< $(OFLAGS)
64
65
66
67  ################################### DEFINITIONS ###############$*---'##############
68
69  # Project, system, and application level definitions
70
71  PROJ            = mra
72  APPSYS          = app
73  COMSYS          = com
```



```
74  ICNSYS          = icn
75  MPUSYS          = mpu
76
77  COMLIB          = \$(PROJ)\lib
78  COMSRC          = \$(PROJ)\$(COMSYS)\src
79  COMBIN31        = \$(PROJ)\$(COMSYS)\bin\8031
80  COMBIN152       = \$(PROJ)\$(COMSYS)\bin\80152
81  COMBINMS        = \$(PROJ)\$(COMSYS)\bin\msdos
82  COMBINSBC8      = \$(PROJ)\$(COMSYS)\bin\sbc8
83
84  # Common subsystem level source directories
85
86  HDRSRC          = $(COMSRC)\hdr
87  LCSSRC          = $(COMSRC)\lcs
88
89  # Common subsystem global include and compilation units
90
91  SYSDEFS         = $(HDRSRC)\sysdefs.h
92
93  LCS             = $(COMBIN152)\lcd.obj          $(COMBIN152)\lci.obj    \
94                    $(COMBIN31)\lcd.obj           $(COMBIN31)\lci.obj     \
95                    $(COMBINMS)\lcd.obj           $(COMBINMS)\lci.obj
96
97
98
99  ################################### TARGETS #####################################
100
101 lcs             : $(LCS)
102
103 lib             : $(LCS)
104                   -lib51 delete $(COMLIB)\mra_1521.lib (lcd, lci)
105                   -lib51 add $(COMBIN152)\lcd.obj to $(COMLIB)\mra_1521.lib
106                   -lib51 add $(COMBIN152)\lci.obj to $(COMLIB)\mra_1521.lib
107                   -lib51 delete $(COMLIB)\mra_311.lib (lcd, lci)
108                   -lib51 add $(COMBIN31)\lcd.obj to $(COMLIB)\mra_311.lib
109                   -lib51 add $(COMBIN31)\lci.obj to $(COMLIB)\mra_311.lib
110                   -lib $(COMLIB)\mra_mss.lib -lcd.obj+$(COMBINMS)\lcd.obj;
111                   -lib $(COMLIB)\mra_msl.lib -lcd.obj+$(COMBINMS)\lcd.obj;
112                   -lib $(COMLIB)\mra_mss.lib -lci.obj+$(COMBINMS)\lci.obj;
113                   -lib $(COMLIB)\mra_msl.lib -lci.obj+$(COMBINMS)\lci.obj;
114                   touch lib
115
116 print:            $(LCS)
117                   %-a2ps -nf lcd.h | post
118                   %-a2ps -nf lcd.c | post
119                   %-a2ps -nf lci.h | post
120                   %-a2ps -nf lci.c | post
121                   touch print
122
123
124
125 ############################## COM LCS DEPENDENCIES ##############################
126
127 # COM Local Communications Device module dependencies for 80152, 8031, MS-DOS
128
129 LCD = $(SYSDEFS)        $(LCSSRC)\bmf.h         $(LCSSRC)\bmf.c         \
130       $(LCSSRC)\lcc.h   $(LCSSRC)\lcc.c         $(LCSSRC)\lcd.h         \
131       $(LCSSRC)\lcd.c
132
133 $(COMBIN152)\lcd.obj            : $(LCD)
134   $(CC) $(LCSSRC)\$*.c $(CFLAGS) df(I80152) pr($(LCSSRC)\$*.152) oj($(COMBIN152
135
136 $(COMBIN31)\lcd.obj             : $(LCD)
137   $(CC) $(LCSSRC)\$*.c $(CFLAGS) df(I8031)  pr($(LCSSRC)\$*.31)  oj($(COMBIN31)
138
139 $(COMBINMS)\lcd.obj             : $(LCD)
140   $(MSC) $(MSCFLAGS) /DIBMAT /Fs$(LCSSRC)\$*.at /Fo$(COMBINMS)\$* $(LCSSRC)\$*.
141
142
143 # COM Local Communications Interface module dependencies for 8031, MS-DOS
144
145 LCI = $(SYSDEFS) $(LCSSRC)\lcd.h $(LCSSRC)\lci.h $(LCSSRC)\lci.c
146
```

```
147   $(COMBIN152)\lci.obj              : $(LCI)
148     $(CC) $(LCSSRC)\$*.c $(CFLAGS) df(180152) pr($(LCSSRC)\$*.152) oj($(COMBIN152
149
150   $(COMBIN31)\lci.obj               : $(LCI)
151     $(CC) $(LCSSRC)\$*.c $(CFLAGS) df(18031) pr($(LCSSRC)\$*.31) oj($(COMBIN31)\$
152
153   $(COMBINMS)\lci.obj               : $(LCI)
154     $(MSC) $(MSCFLAGS) /DIBMAT /Fa$(LCSSRC)\$*.at /Fo$(COMBINMS)\$* $(LCSSRC)\$*.
```

```
1   /*****************************************************************************\
2   *                                  BMF.H                                      *
3   ******************************************************************************
4   *                                                                            *
5   * CPCI:          IED90-MRA-COM-LCS-BMF-H-R0C0                                *
6   *                                                                            *
7   * Description:   Frame Buffer Management variables and definitions.          *
8   *                Contains constant declarations and type definitions for the *
9   *                circular packet buffer management functions. Used by the    *
10  *                LCS communications device handler module.                   *
11  *                                                                            *
12  *                Module BMF exports the following types/functions:           *
13  *                                                                            *
14  *                typedef buffer;                                             *
15  *                                                                            *
16  *                buf_full();                                                 *
17  *                buf_empty();                                                *
18  *                buf_clear();                                                *
19  *                buf_insert();                                               *
20  *                buf_remove();                                               *
21  *                                                                            *
22  * Notes:         1) This file is included by LCD.C.                          *
23  *                2) Module BMF requires type definitions from LCD.H.         *
24  *                                                                            *
25  * Edit History: 06/28/90 - Written by Robin T. Laird.                        *
26  *                                                                            *
27  \*****************************************************************************/
28
29  /* Private Data Structures:                                                  */
30
31  #ifndef BMF_MODULE_CODE
32  #define BMF_MODULE_CODE          3100
33
34  #define ERR_FULL_BUFFER          1+BMF_MODULE_CODE
35  #define ERR_EMPTY_BUFFER         2+BMF_MODULE_CODE
36
37  #define ERR_BOP_NOT_FOUND        3+BMF_MODULE_CODE
38  #define ERR_CRC_INVALID          4+BMF_MODULE_CODE
39  #define ERR_INVALID_LENGTH       5+BMF_MODULE_CODE
40
41  /* MAX_BUFFER_SIZE defines the number of bytes/buffer.                      */
42
43  #define MAX_BUFFER_SIZE          1024
44
45  /* Circular buffer (queue) to hold incoming/outgoing data packets.          */
46  /* Must be initialized using buf_clear().                                   */
47
48  typedef struct { byte item[MAX_BUFFER_SIZE];
49                   word front;
50                   word rear;
51                   byte empty;
52                   byte full;
53                 } buffer;
54
55  /* Private Functions:                                                       */
56
57  static int buf_full(buffer *b);
58  static int buf_empty(buffer *b);
59  static void buf_clear(buffer *b);
60  static void buf_insert(buffer *b, lcd_packet p);
61  static void buf_remove(buffer *b, lcd_packet p);
62
63  #endif
```

```
1   /*****************************************************************************\
2   *                                  BMF.C                                      *
3   *****************************************************************************
4   *                                                                            *
5   * CPCI:          IED90-MRA-COM-LCS-BMF-C-R0C2                                *
6   *                                                                            *
7   * Description:   Frame Buffer Management functions.                          *
8   *                Contains functions for initializing and managing the        *
9   *                circular packet buffers for the LCS local communications    *
10  *                device handler.                                             *
11  *                                                                            *
12  *                Module BMF exports the following functions:                 *
13  *                                                                            *
14  *                buf_front() macro;                                          *
15  *                buf_rear()  macro;                                          *
16  *                buf_inc()   macro;                                          *
17  *                buf_full();                                                 *
18  *                buf_empty();                                                *
19  *                buf_clear();                                                *
20  *                buf_insert();                                               *
21  *                buf_remove();                                               *
22  *                                                                            *
23  * Notes:         1) This module is NOT a stand-alone compilation unit.       *
24  *                   It is included by the module LCD.C and is compiled there.*
25  *                   It is assumed that the file LCD.H is included before it. *
26  *                   Note that all of the functions herein are static.        *
27  *                                                                            *
28  * Edit History: 08/14/90 - Written by Richard P. Smurlo and Robin T. Laird.  *
29  *                                                                            *
30  \*****************************************************************************/
31
32  /* Private Variables:                                                       */
33
34  /* Declarations for the module circular receive and transmit buffers.       */
35  /* These are global to the LCC.C module and to the LCD.C module.            */
36
37  static XDATA buffer xmt_buffer;
38  static XDATA buffer rcv_buffer;
39
40  /* The packet field lengths below are given in number of bytes.             */
41
42  #define BOP_LENGTH              3               /* Beginning-of-packet len. */
43  #define ADDR_LENGTH             1               /* Source address len.      */
44  #define LENGTH_LENGTH           1               /* Packet length len.       */
45  #define CRC_LENGTH              2               /* Checksum (CRC) len.      */
46
47  /* Packet overhead is number of bytes added to packet before it is sent.    */
48  /* This includes the beginning-of-packet (BOP) and the CRC.                 */
49  /* The BMF software adds the BOP flag (3 bytes) and the CRC (2 bytes).      */
50
51  #define PACKET_OVERHEAD         (BOP_LENGTH+CRC_LENGTH)
52
53  /* The minimum number of bytes for a valid packet (as seen from this level) */
54  /* is the length of the packet overhead plus the length of the (source)     */
55  /* address field and the length field.                                      */
56
57  #define MIN_BYTES_IN_PACKET     (PACKET_OVERHEAD+ADDR_LENGTH+LENGTH_LENGTH)
58
59  /* BOP is defined as a "unique" sequence of bits that precedes each packet. */
60
61  static byte BOP[] = {0xAA, 0xAA, 0xAA};         /* 1010 1010 1010           */
62
63
64  /*****************************************************************************\
65  *                                buf_front                                   *
66  *****************************************************************************
67  *                                                                            *
68  * Function:      Macro that returns a pointer to the front (current)         *
69  *                element in the parameter buffer. Used to obtain the         *
70  *                address of the next packet to be received/transmitted.      *
71  *                                                                            *
72  * Input:         buf_front(                                                  *
73  *                  buffer b; buffer structure (NOT a pointer to it).         *
```



```
74  *                );                                                          *
75  *                                                                            *
76  * Output:        Pointer to front (current) packet in the buffer.            *
77  *                                                                            *
78  * Globals:       None.                                                       *
79  *                                                                            *
80  * Edit History: 07/08/90 - Written by Robin T. Laird.                        *
81  *                                                                            *
82  \*****************************************************************************/
83
84  #define buf_front(b) (&b.item[b.front])
85
86
87  /*****************************************************************************\
88  *                                buf_rear                                    *
89  *****************************************************************************
90  *                                                                            *
91  * Function:      Macro that returns a pointer to the rear (last)             *
92  *                element in the parameter buffer. Used to obtain the         *
93  *                address of the next packet to be received/transmitted.      *
94  *                                                                            *
95  * Input:         buf_rear(                                                   *
96  *                  buffer b; buffer structure (NOT a pointer to it).         *
97  *                );                                                          *
98  *                                                                            *
99  * Output:        Pointer to rear (last) packet in the buffer.                *
100 *                                                                            *
101 * Globals:       None.                                                       *
102 *                                                                            *
103 * Edit History: 07/10/90 - Written by Robin T. Laird.                        *
104 *                                                                            *
105 \*****************************************************************************/
106
107 #define buf_rear(b) (&b.item[b.rear])
108
109
110 /*****************************************************************************\
111 *                                buf_inc                                     *
112 *****************************************************************************
113 *                                                                            *
114 * Function:      Macro that increments either the front or rear index         *
115 *                of a circular buffer of max size MAX_BUFFER_SIZE.           *
116 *                Should be used with care, i.e., not nested within a         *
117 *                compound statement like if ... then ... else.               *
118 *                                                                            *
119 * Input:         buf_inc(                                                    *
120 *                  word x; index to increment.                               *
121 *                );                                                          *
122 *                                                                            *
123 * Output:        Nothing.                                                    *
124 *                                                                            *
125 * Globals:       None.                                                       *
126 *                                                                            *
127 * Edit History: 12/04/90 - Written by Robin T. Laird.                        *
128 *                                                                            *
129 \*****************************************************************************/
130
131 #define buf_inc(x) (x = ((x == MAX_BUFFER_SIZE-1) ? 0 : x+1))
132
133
134 /*****************************************************************************\
135 *                                buf_full                                    *
136 *****************************************************************************
137 *                                                                            *
138 * Function:      Function that returns the boolean of whether or not the     *
139 *                parameter buffer is full (TRUE if so, FALSE if not).        *
140 *                                                                            *
141 * Input:         buf_full(                                                   *
142 *                  buffer *b; pointer to buffer structure.                   *
143 *                );                                                          *
144 *                                                                            *
145 * Output:        Integer, TRUE if buffer FULL, FALSE if buffer not FULL.     *
146 *                                                                            *
```

```
147   * Globals:       None.                                                  *
148   *                                                                        *
149   * Edit History: 07/08/90 - Written by Robin T. Laird.                   *
150   *                                                                        *
151   \************************************************************************/
152
153   static int buf_full(b)
154   buffer *b;
155   {
156     lcd_error = AOK;                        /* Assume function successful...    */
157     return(b->full);
158   }
159
160
161   /************************************************************************\
162   *                                buf_empty                               *
163   ************************************************************************
164   *                                                                        *
165   * Function:      Function that returns the boolean of whether or not the *
166   *                parameter buffer is empty (TRUE if so, FALSE if not).   *
167   *                Operation depends upon the parameter buffer as follows: *
168   *                                                                        *
169   *                If buffer b == xmt_buffer then just returns empty flag. *
170   *                If buffer b == rcv_buffer then returns whether packet avail. *
171   *                                                                        *
172   *                The receive buffer is considered empty if either 1) there *
173   *                are not enough bytes to determine the length of the incoming *
174   *                packet, or 2) the number of bytes received is less than the *
175   *                number indicated by the packet length byte (which HAS been *
176   *                received).                                              *
177   *                                                                        *
178   *                Note that this routine advances the front buffer pointer to *
179   *                the beginning of a valid packet. It MUST be called before a *
180   *                packet is actually removed from the buffer.             *
181   *                                                                        *
182   * Input:         buf_empty(                                              *
183   *                  buffer *b; pointer to buffer structure.               *
184   *                );                                                      *
185   *                                                                        *
186   * Output:        Integer, TRUE if buffer EMPTY, FALSE if buffer not EMPTY. *
187   *                                                                        *
188   * Globals:       None.                                                   *
189   *                                                                        *
190   * Edit History: 12/10/90 - Written by Richard P. Smurlo and Robin T. Laird. *
191   *               03/08/91 - Fixed bug when first packet bad, Robin T. Laird. *
192   *                                                                        *
193   \************************************************************************/
194
195   static int buf_empty(b)
196   buffer *b;
197   {
198     int  bopix;                             /* BOP byte sequence index.         */
199     int  bfull;                             /* Buffer full flag.                */
200     word bytes_in_buff;                     /* Num bytes currently in buffer.   */
201     word packet_length;                     /* Actual calculated packet length. */
202     word bfront, brear;                     /* Static buffer front and rear.    */
203
204     lcd_error = AOK;                        /* Assume function successful...    */
205
206     /* Just return structure flag for transmit buffer.                        */
207     /* Otherwise, determine if there are enough bytes for an entire packet.   */
208
209     if (b == &xmt_buffer)
210       return(xmt_buffer.empty);
211     else
212     {
213       /* Quick check to see if buffer REALLY empty.                           */
214
215       if ((b->front == b->rear) && !b->full) return(TRUE);
216
217       /* Set local buffer front, rear, full variables.                        */
218       /* Interrupts are happening, we don't want our vars changing as we go.  */
219
```



```
220       bfront = b->front;
221       brear  = b->rear;
222       bfull  = b->full;
223
224       /* Calculate the number of bytes currently in the buffer.               */
225       /* Num bytes in buffer must be > minimum bytes for a valid packet.      */
226
227       if (bfull)
228         bytes_in_buff = MAX_BUFFER_SIZE;
229       else if (bfront < brear)
230         bytes_in_buff = brear - bfront;
231       else
232         bytes_in_buff = brear + MAX_BUFFER_SIZE - bfront;
233
234       /* If there aren't enough bytes, report AOK, return buffer empty.      */
235
236       if (bytes_in_buff < MIN_BYTES_IN_PACKET)
237       {
238         return(TRUE);
239       }
240       else
241       {
242         /* We have enough bytes to VERIFY if we have a valid packet.          */
243         /* Find the BOP sequence in the buffer (may have a bad packet).       */
244         /* If we hit buffer end, report BOP not found, return buffer empty.   */
245         /* Also, buffer is actually empty so make front=rear and full=FALSE.  */
246
247         bopix = 0;
248         while (bopix < BOP_LENGTH)
249         {
250           if (b->item[bfront] == BOP[bopix])
251             bopix++;
252           else
253             bopix = 0;
254           buf_inc(bfront);
255           if ((bfront == brear) && (bopix < BOP_LENGTH))
256           {
257             lcd_error = ERR_BOP_NOT_FOUND;
258             b->front = bfront;
259             b->full  = FALSE;
260             return(TRUE);
261           }
262         }
263
264         /* Must have a valid BOP and bfront points to byte after BOP.         */
265         /* Move actual buffer front (b->front) to beginning of BOP.           */
266
267         b->front = bfront;
268         for (bopix = 0; bopix < BOP_LENGTH; bopix++)
269         {
270           if (b->front == 0)
271             b->front = MAX_BUFFER_SIZE - 1;
272           else
273             b->front--;
274         }
275
276         /* Now recalculate number of bytes in buffer to see if packet valid.  */
277         /* Must add BOP_LENGTH since bfront points to byte past BOP.          */
278
279         if (bfront < brear)
280           bytes_in_buff = brear - bfront + BOP_LENGTH;
281         else
282           bytes_in_buff = brear + MAX_BUFFER_SIZE - bfront + BOP_LENGTH;
283
284         /* If there aren't enough bytes, report AOK, return buffer empty.     */
285
286         if (bytes_in_buff < MIN_BYTES_IN_PACKET)
287         {
288           return(TRUE);
289         }
290         else
291         {
292           /* If we reach here...                                              */
```

```
293           /*   b->front points to beginning of BOP.                    */
294           /*   bfront points to first byte in packet (past BOP).       */
295           /*   bytes_in_buff holds num bytes currently in buffer.      */
296           /*   There are enough bytes in buffer to VERIFY valid packet.*/
297
298           /* Retrieve length byte of packet.                           */
299           /* Length byte is at LCD_LEN_POS bytes past first byte in packet.*/
300           /* Routine FAILS if packet length byte is invalid (sometimes).*/
301           /* Result depends on if bad packet length > num bytes in buffer.*/
302
303           if (bfront < MAX_BUFFER_SIZE - LCD_LEN_POS)
304             packet_length = b->item[bfront + LCD_LEN_POS];
305           else
306             packet_length = b->item[bfront + LCD_LEN_POS - MAX_BUFFER_SIZE];
307
308           /* Packet INVALID if num bytes in buff < packet_length + overhead.*/
309           /* Return buffer empty if so, otherwise return buffer NOT empty.*/
310
311           if (bytes_in_buff < packet_length + PACKET_OVERHEAD)
312             return(TRUE);
313           else
314             return(FALSE);
315         }
316       }
317     }
318   }
319
320
321   /*****************************************************************************\
322   *                                buf_clear                                  *
323   ******************************************************************************
324   *                                                                           *
325   * Function:     Initializes the parameter buffer and clears its contents.   *
326   *               The front and rear pointers are reset and the boolean       *
327   *               state flags (i.e., empty, full) are set accordingly.        *
328   *                                                                           *
329   * Input:        buf_clear(                                                  *
330   *                 buffer *b; pointer to buffer structure to clear.          *
331   *               );                                                          *
332   *                                                                           *
333   * Output:       Nothing.                                                    *
334   *                                                                           *
335   * Globals:      lcd_error : module LCD.C                                    *
336   *                                                                           *
337   * Edit History: 07/09/90 - Written by Robin T. Laird.                       *
338   *                                                                           *
339   \*****************************************************************************/
340
341   static void buf_clear(b)
342   buffer *b;
343   {
344     word i;
345
346     lcd_error = AOK;                        /* Assume function successful...*/
347
348     /* Process each of the packets in the buffer.                          */
349     /* Set all bytes in packet to 0.                                       */
350
351     for (i = 0; i < MAX_BUFFER_SIZE; i++)
352     {
353       b->item[i] = 0x00;
354     }
355
356     /* Set the front and rear indexes equal to indicate empty buffer.      */
357     /* Set the empty and full flags as appropriate.                        */
358
359     b->front = b->rear = 0;
360     b->empty = TRUE;
361     b->full  = FALSE;
362   }
363
364
365   /*****************************************************************************\
```



```
366   *                                buf_insert                                 *
367   ******************************************************************************
368   *                                                                           *
369   * Function:     Inserts a packet into the parameter buffer if room.         *
370   *               An error is returned if the buffer is already full.         *
371   *               Inserts beginning-of-packet header and CRC trailer.         *
372   *                                                                           *
373   * Input:        buf_insert(                                                 *
374   *                 buffer *b;    pointer to buffer structure to clear.       *
375   *                 lcd_packet p; packet to be inserted.                      *
376   *               );                                                          *
377   *                                                                           *
378   * Output:       Nothing.                                                    *
379   *                                                                           *
380   * Globals:      lcd_error : module LCD.C                                    *
381   *                                                                           *
382   * Edit History: 07/09/90 - Written by Robin T. Laird.                       *
383   *                                                                           *
384   \*****************************************************************************/
385
386   static void buf_insert(b, p)
387   buffer *b;
388   lcd_packet p;
389   {
390     word i, crc, length, size;
391
392     lcd_error = AOK;                        /* Assume function successful...*/
393
394     /* Make sure buffer isn't already full.                                */
395
396     if (b->full)
397     {
398       lcd_error = ERR_FULL_BUFFER;
399       return;
400     }
401
402     /* Calculate size and remaining space in buffer to see if packet will fit.*/
403     /* If there's not enough room, report buffer full and return.          */
404     /* Length of packet is stored at LCD_LEN_POS in packet data.           */
405
406     if (b->front <= b->rear)
407       size = b->rear - b->front;
408     else
409       size = b->rear + MAX_BUFFER_SIZE - b->front + 1;
410
411     length = p[LCD_LEN_POS];
412     if (length + PACKET_OVERHEAD > MAX_BUFFER_SIZE - size)
413     {
414       lcd_error = ERR_FULL_BUFFER;
415       return;
416     }
417
418     /* Insert beginning of packet (BOP) header into buffer.                */
419
420     for (i = 0; i < BOP_LENGTH; i++)
421     {
422       b->item[b->rear] = BOP[i];
423       buf_inc(b->rear);
424     }
425
426     /* Copy element into buffer and set appropriate structure fields.      */
427     /* Adjust rear index (MAX_BUFFER_SIZE-1 is last element in buffer).    */
428
429     for (i = 0; i < length; i++)
430     {
431       b->item[b->rear] = p[i];
432       buf_inc(b->rear);
433     }
434
435     /* Calculate CRC checksum and insert into buffer.                      */
436
437     crc = lcc_crc16(p, length);
438
```

```
439     b->item[b->rear] = crc >> 8;
440     buf_inc(b->rear);
441     b->item[b->rear] = crc & 0x00FF;
442     buf_inc(b->rear);
443
444     /* Check and see if we've filled the buffer (set full flag if so).      */
445     /* Set empty flag to FALSE since we just inserted.                      */
446
447     if ((b->front == b->rear) && !b->empty) b->full = TRUE;
448     b->empty = FALSE;
449   }
450
451
452   /*************************************************************************\
453   *                              buf_remove                                 *
454   *************************************************************************
455   *                                                                        *
456   * Function:     Removes a packet from the parameter buffer if available. *
457   *               An error is returned if the buffer is already empty.     *
458   *               The beginning-of-packet header and CRC trailer are removed *
459   *               before the data packet is returned.                      *
460   *                                                                        *
461   * Input:        buf_remove(                                              *
462   *                 buffer *b;    pointer to buffer structure to clear.    *
463   *                 lcd_packet p; packet to be removed.                    *
464   *               );                                                       *
465   *                                                                        *
466   * Output:       Nothing.                                                 *
467   *                                                                        *
468   * Globals:      lcd_error : module LCD.C                                 *
469   *                                                                        *
470   * Edit History: 07/09/90 - Written by Robin T. Laird.                    *
471   *               03/08/91 - Compatible with new buf_empty(), Robin T. Laird. *
472   *                                                                        *
473   \*************************************************************************/
474
475   static void buf_remove(b, p)
476   buffer *b;
477   lcd_packet p;
478   {
479     word i, length;
480     word crc_calc, crc_rcvd, crc_hi, crc_lo;
481
482     lcd_error = AOK;                    /* Assume function successful...    */
483
484     /* Make sure buffer isn't already empty.                                */
485     /* After buf_empty() call, b->front will point to beginning of BOP.     */
486
487     if (buf_empty(b))
488     {
489       lcd_error = ERR_EMPTY_BUFFER;
490       return;
491     }
492
493     /* Valid packet in buffer (buffer not empty), proceed to remove.        */
494     /* Discard and skip past BOP.                                           */
495
496     for (i = 0; i < BOP_LENGTH; i++) buf_inc(b->front);
497
498     /* Calculate the length of the packet.                                  */
499
500     if (b->front < MAX_BUFFER_SIZE - LCD_LEN_POS)
501       length = b->item[b->front + LCD_LEN_POS];
502     else
503       length = b->item[b->front + LCD_LEN_POS - MAX_BUFFER_SIZE];
504
505     /* Copy length number of bytes.                                         */
506
507     for (i = 0; i < length; i++)
508     {
509        p[i] = b->item[b->front];
510        buf_inc(b->front);
511     }
```



```
512
513     /* Strip CRC off the returned packet.                                   */
514
515     crc_hi = b->item[b->front];
516     buf_inc(b->front);
517
518     crc_lo = b->item[b->front];
519     buf_inc(b->front);
520
521     /* Calculate CRC based on data received.                                */
522     /* Compare calculated CRC against CRC received.                         */
523     /* If they don't match, report error as invalid CRC.                    */
524
525     crc_rcvd = (crc_hi << 8) | crc_lo;
526     crc_calc = lcc_crc16(p, length);
527
528     if (crc_calc != crc_rcvd)
529     {
530       lcd_error = ERR_CRC_INVALID;
531       return;
532     }
533
534     /* Buffer can't be full since we just removed a packet.                 */
535
536     b->full = FALSE;
537
538     /* Check and see if we've depleted the buffer (set empty flag if so).   */
539
540     if (b->front == b->rear) b->empty = TRUE;
541   }
```

```
1   /*****************************************************************************\
2   *                                  LCC.H                                      *
3   ******************************************************************************
4   *                                                                            *
5   * CPCI:          IED90-MRA-COM-LCS-LCC-H-R0C0                                *
6   *                                                                            *
7   * Description:   Local Communications Channel (LCC) variables and functions. *
8   *                Contains constant function parameter and hardware register  *
9   *                declarations for initializatin and control of the Intel     *
10  *                8031 LCC. Implements the low-level data link layer portion  *
11  *                of the MRA LCS communications device handler functions.     *
12  *                                                                            *
13  *                Module LCC exports the following functions:                 *
14  *                                                                            *
15  *                lcc_mode();                                                 *
16  *                lcc_baud();                                                 *
17  *                lcc_crc16();                                                *
18  *                lcc_sys_init();                                             *
19  *                lcc_set_rcv_dst();                                          *
20  *                lcc_set_xmt_src();                                          *
21  *                lcc_start_rcv();                                            *
22  *                lcc_start_xmt();                                            *
23  *                lcc_stop_rcv();                                             *
24  *                lcc_stop_xmt();                                             *
25  *                lcc_enable_interrupts();                                    *
26  *                lcc_sio_int()  interrupt;                                   *
27  *                lcc_sio1_int() interrupt;                                   *
28  *                lcc_sio2_int() interrupt;                                   *
29  *                lcc_sio3_int() interrupt;                                   *
30  *                lcc_sio4_int() interrupt;                                   *
31  *                lcc_pio_Int()  interrupt;                                   *
32  *                                                                            *
33  * Notes:         1) See the Intel 8-Bit Embedded Controller Handbook for     *
34  *                   more information (No. 270645-002, pp. 7-11 - 7-21).      *
35  *                                                                            *
36  * Edit History: 08/14/90 - Written by Robin T. Laird and Richard P. Smurlo.  *
37  *                                                                            *
38  \*****************************************************************************/
39
40  /* Private Data Structures:                                                  */
41
42  #ifndef LCC_MODULE_CODE
43  #define LCC_MODULE_CODE          3200
44
45  #define ERR_IO_MODE              1+LCC_MODULE_CODE
46  #define ERR_BAUD_RATE            2+LCC_MODULE_CODE
47  #define ERR_RFIFO_NOT_CLEAR      3+LCD_MODULE_CODE
48
49  /* I/O mode settings (specifies serial or parallel channel operation).      */
50
51  #define LCC_SIO_MODE             0
52  #define LCC_PIO_MODE             1
53
54  /* Serial I/O Constants:                                                    */
55
56  /* I80152 signifies baud rate settings for 14.7456 MHz CPU.                 */
57  /* I8031  signifies baud rate settings for 11.0592 MHz CPU.                 */
58  /* IBMAT signifies settings for the IBM-AT (8250).                          */
59
60  #if defined(I80152)
61  #define MAX_BAUD_RATE            0xFF    /* 38.4 KBPS (SMOD=0).           */
62  #define BR19200                  0xFE
63  #define BR9600                   0xFC
64  #define BR4800                   0xF8
65  #define BR2400                   0xF0
66  #define BR1200                   0xE0
67  #define BR600                    0xC0
68  #define BR300                    0x80
69
70  #elif defined(I8031)
71  #define MAX_BAUD_RATE            0xFF    /* 57.6 KBPS (SMOD=1).           */
72  #define BR19200                  0xFD
73  #define BR9600                   0xFA
```



```
74  #define BR4800                   0xF4
75  #define BR2400                   0xE8
76  #define BR1200                   0xD0
77  #define BR600                    0xA0
78  #define BR300                    0x40
79
80  #elif defined(IBMAT)
81  #define BR38400                  3       /* Baud rate divisor settings.  */
82  #define BR19200                  6
83  #define BR9600                   12
84  #define BR4800                   24
85  #define BR2400                   48
86  #define BR1200                   96
87  #define BR600                    192
88  #define BR300                    384
89
90  #define PEVEN                    0x18    /* Parity settings.             */
91  #define PODD                     0x08
92  #define PNONE                    0x00
93
94  #define WL5                      0x00    /* Word length settings.        */
95  #define WL6                      0x01
96  #define WL7                      0x02
97  #define WL8                      0x03
98
99  #define SB1                      0x00    /* Stop bit settings.           */
100 #define SB2                      0x04
101
102 /* COM port base addresses.                                                 */
103
104 #define COM1                     0x3F0   /* 3F8-3FF.                     */
105 #define COM2                     0x2F0   /* 2F8-2FF.                     */
106 #define COM3                     0x3E0   /* 3E8-3EF.                     */
107 #define COM4                     0x2E0   /* 2E8-2EF.                     */
108
109 /* Global port and line control variables for customization.                */
110
111 extern unsigned int  lcc_com_port;
112 extern unsigned int  lcc_baud_rate;
113 extern unsigned char lcc_tty_control;
114 #endif
115
116 /* Parallel I/O Constants:                                                  */
117
118 /* Data direction (PDIR) definitions: 0 = input, 1 = output.                */
119
120 #define PIO_INPUT                0
121 #define PIO_OUTPUT               1
122
123 /* Private Functions:                                                       */
124
125 static int  lcc_mode(void);
126 static int  lcc_baud(void);
127 static void lcc_set_rcv_dst(lcd_packet p, word length);
128 static void lcc_set_xmt_src(lcd_packet p, word length);
129 static word lcc_crc16(lcd_packet p, word length);
130 static void lcc_sys_init(int mode, int baud_rate);
131 static void lcc_start_rcv(void);
132 static void lcc_start_xmt(void);
133 static void lcc_stop_rcv(void);
134 static void lcc_stop_xmt(void);
135 static void lcc_enable_interrupts();
136 #if defined(IBMAT)
137 static void interrupt far lcc_sio1_int();
138 static void interrupt far lcc_sio2_int();
139 static void interrupt far lcc_sio3_int();
140 static void interrupt far lcc_sio4_int();
141 #endif
142
143 #endif
```

```
1   /*****************************************************************************\
2   *                                  LCC.C                                      *
3   *****************************************************************************
4   *                                                                            *
5   * CPCI:          IED90-MRA-COM-LCS-LCC-C-R0C0                                *
6   *                                                                            *
7   * Description:   Local Communications Channel (LCC) functions.               *
8   *                Implements the MRA low-level data link layer functions      *
9   *                for the LCS local communications device handler.            *
10  *                Currently supports the 8031 Local Serial Channel (LSC), the *
11  *                the 80C152 Local Parallel Channel (LPC - I/O ports P4/P5),  *
12  *                and the serial communications ports of the IBM-AT (8250).   *
13  *                                                                            *
14  *                Module LCC exports the following functions:                 *
15  *                                                                            *
16  *                xptr_lo_offset() macro;                                     *
17  *                xptr_hi_offset() macro;                                     *
18  *                lcc_mode();                                                 *
19  *                lcc_baud();                                                 *
20  *                lcc_crc16();                                                *
21  *                lcc_sys_init();                                             *
22  *                lcc_set_rcv_dst();                                          *
23  *                lcc_set_xmt_src();                                          *
24  *                lcc_start_rcv();                                            *
25  *                lcc_start_xmt();                                            *
26  *                lcc_stop_rcv();                                             *
27  *                lcc_stop_xmt();                                             *
28  *                lcc_enable_interrupts();                                    *
29  *                lcc_sio_int()  interrupt;                                   *
30  *                lcc_sio1_int() interrupt;                                   *
31  *                lcc_sio2_int() interrupt;                                   *
32  *                lcc_sio3_int() interrupt;                                   *
33  *                lcc_sio4_int() interrupt;                                   *
34  *                lcc_pio_int()  interrupt;                                   *
35  *                                                                            *
36  * Notes:         1) This module is NOT a stand-alone compilation unit.       *
37  *                   It is included by the module LCD.C and is compiled there.*
38  *                   It is assumed that the file LCD.H is included before it. *
39  *                   Note that all of the functions herein are static.        *
40  *                                                                            *
41  * Edit History: 07/10/90 - Written by Richard P. Smurlo and Robin T. Laird.  *
42  *                                                                            *
43  \*****************************************************************************/
44
45  #if defined(IBMAT)
46  #include <dos.h>                        /* DOS IBM-AT service definitions.  */
47  #else
48  #include <reg152.h>                     /* 8031 and 80152 definitions.      */
49  #endif
50
51  /* Private Variables:                                                        */
52
53  /* Variable that indicates operational mode (serial or parallel).           */
54  /* Set by the lcc_sys_init() routine (default is serial mode).               */
55
56  static XDATA int lcc_io_mode = LCC_SIO_MODE;
57
58  /* Communications port definitions for MPU (IBM-AT 8250).                    */
59  /* COM port selection and settings are done in MAIN.C and passed here.       */
60
61  #if defined(IBMAT)
62
63  /* "8250" communications controller (CC) register address offsets.           */
64
65  #define CC_THR                  0x08    /* Transmit holding reg (write).    */
66  #define CC_RBR                  0x08    /* Receiver buffer reg (read).      */
67  #define CC_DLSB                 0x08    /* Divisor latch LSB.               */
68  #define CC_DMSB                 0x09    /* Divisor latch MSB.               */
69  #define CC_IER                  0x09    /* Interrupt enable register.       */
70  #define CC_IIR                  0x0A    /* Interrupt identification reg.    */
71  #define CC_LCR                  0x0B    /* Line control register            */
72  #define CC_MCR                  0x0C    /* Modem control register.          */
73  #define CC_LSR                  0x0D    /* Line status register.            */
```



```
74  #define CC_MSR                  0x0E    /* Modem status register.           */
75
76  #define CC_DATA_READY           0x01    /* LSR data ready bit.              */
77  #define CC_RECEIVE_ERRS         0x1E    /* LSR error bits mask.             */
78  #define CC_TRANSMIT_HOLD_EMPTY  0x20    /* LSR xmtr holding register empty. */
79
80  #define CC_DLAB0                0x00    /* DLAB set low (0).                */
81  #define CC_DLAB1                0x80    /* DLAB set high (1).               */
82
83  #define CC_DISABLE_INT          0x00    /* Disable all interrupts.          */
84  #define CC_ENABLE_RCV           0x01    /* Enable receive interrupts.       */
85  #define CC_ENABLE_XMT           0x02    /* Enable transmit interrupts.      */
86  #define CC_ENABLE_STAT          0x04    /* Enable line status interrupts.   */
87  #define CC_MODEM_BUS_ENABLE     0x0B    /* DTR, RTS, and OUT2 active.       */
88
89  /* "8250" interrupt source numbers (in order of priority from high to low). */
90
91  #define LINE_STATUS_ERR_INT     0x06
92  #define RCV_DATA_AVAIL_INT      0x04
93  #define XMT_EMPTY_INT           0x02
94  #define MODEM_INT               0x00
95
96  /* 8259 interrupt controller (IC) register (absolute) addresses.            */
97
98  #define IC_OCW2                 0x20    /* Operation control word 2.        */
99  #define IC_OCW1                 0x21    /* Operation control word 1.        */
100
101 #define IC_ENABLE_IRQ3          0xF7    /* Enable ints on IRQ3.             */
102 #define IC_ENABLE_IRQ4          0xEF    /* Enable ints on IRQ4.             */
103 #define IC_EOI                  0x20    /* Non-specific end-of-interrupt.   */
104
105 /* Globals that can be changed by user as desired BEFORE call to lci_init(). */
106
107 unsigned int  lcc_com_port    = COM1;
108 unsigned int  lcc_baud_rate   = BR19200;
109 unsigned char lcc_tty_control = (PNONE | WLB | SB1);
110 unsigned char lcc_ic_ocw1;
111 void (interrupt far *lcc_irq3_vect)();
112 void (interrupt far *lcc_irq4_vect)();
113
114 #else
115
116 /* Switch select bits for the ICN and MPU (80152 and 8031).                 */
117 /* Define I/O mode (parallel/serial select) special function bit variable.  */
118
119 sbit PSSEL = P1^6;
120
121 /* Define baud rate select special function bit variables.                  */
122
123 sbit BRS0  = P1^5;
124 sbit BRS1  = P1^4;
125 sbit BRS2  = P1^3;
126
127 /* Define special function register bit/byte variables for LPC.             */
128
129 #define PIO  P4                 /* Parallel I/O (xmt/rcv) data register.    */
130 sbit PDIR  = P5^3;              /* Parallel data direction (0=IN, 1=OUT).   */
131 sbit RTSO_ = P5^4;              /* Request to send output (active low).     */
132 sbit CTSI_ = P5^5;              /* Clear to send input    (active low).     */
133 sbit RTSI_ = P5^6;              /* Request to send input  (active low).     */
134 sbit CTSO_ = P5^7;              /* Clear to send output   (active low).     */
135
136 #endif
137
138
139 /*****************************************************************************\
140 *                                xptr_lo_offset                              *
141 *****************************************************************************
142 *                                                                            *
143 * Function:     Obtains the low-offset portion of an external data (XDATA)   *
144 *               pointer. Specific to the Franklin 8031 C compiler (V2.4).    *
145 *                                                                            *
146 * Input:        xptr_lo_offset(
```

```
147   *                 xdata void *xp; XDATA pointer.                      *
148   *               );                                                    *
149   *                                                                     *
150   * Output:       Returns the low-offset portion of the external data pointer. *
151   *                                                                     *
152   * Globals:      None.                                                 *
153   *                                                                     *
154   * Edit History: 07/07/90 - Written by Robin T. Laird.                 *
155   *                                                                     *
156   \*********************************************************************/
157
158   #define xptr_lo_offset(xp) (((unsigned int)xp)&0x00FF)
159
160
161   /*********************************************************************\
162   *                          xptr_hi_offset                             *
163   **********************************************************************
164   *                                                                     *
165   * Function:     Obtains the hi-offset portion of an external data (XDATA) *
166   *               pointer. Specific to the Franklin 8031 C compiler (V2.4). *
167   *                                                                     *
168   * Input:        xptr_hi_offset(                                       *
169   *                 xdata void *xp; XDATA pointer.                      *
170   *               );                                                    *
171   *                                                                     *
172   * Output:       Returns the hi-offset portion of the external data pointer. *
173   *                                                                     *
174   * Globals:      None.                                                 *
175   *                                                                     *
176   * Edit History: 07/07/90 - Written by Robin T. Laird.                 *
177   *                                                                     *
178   \*********************************************************************/
179
180   #define xptr_hi_offset(xp) ((((unsigned int)xp)>>8)&0x00FF)
181
182
183   /*********************************************************************\
184   *                             lcc_mode                                *
185   **********************************************************************
186   *                                                                     *
187   * Function:     Determines the I/O mode for external communications.  *
188   *                                                                     *
189   *               For the 80C152:                                       *
190   *                                                                     *
191   *               The mode is read from a switch connected to P1 bit 6 of *
192   *               the 80C152 parallel port. Low (0) selects serial mode, *
193   *               while high (1) selects parallel mode. The mode bit    *
194   *               corresponds to P/S SEL on the ICN schematic.          *
195   *                                                                     *
196   *               For the 8031 (or any other processor):                *
197   *                                                                     *
198   *               The mode is always returned as serial mode selected (0). *
199   *                                                                     *
200   * Input:        lcc_mode();                                           *
201   *                                                                     *
202   * Output:       Returns the local communications mode, 0 for serial mode, *
203   *               1 for parallel mode.                                  *
204   *                                                                     *
205   * Globals:      lcd_error : module LCD.C                              *
206   *               8031 regs : module REG152.H                           *
207   *                                                                     *
208   * Edit History: 08/14/90 - Written by Robin T. Laird.                 *
209   *                                                                     *
210   \*********************************************************************/
211
212   static int lcc_mode()
213   {
214     lcd_error = AOK;                        /* Assume function successful...  */
215
216     #if defined(I80152)
217     return ((int)PSSEL);
218     #else
219     return (LCC_SIO_MODE);
```



```
220     #endif
221   }
222
223
224   /*********************************************************************\
225   *                             lcc_baud                                *
226   **********************************************************************
227   *                                                                     *
228   * Function:     Determines the baud rate for external serial communications. *
229   *                                                                     *
230   *               For the 80C152:                                       *
231   *                                                                     *
232   *               The baud rate is read from a switch connected to P1 of *
233   *               the 80C152 parallel port. Bits 3, 4, and 5 select one of *
234   *               eight available baud rates from 300 baud to 38400 baud. *
235   *               Bit 3 corresponds to BSR2 on the ICN schematic, 4 is BSR1, *
236   *               and bit 5 to BSR0.                                    *
237   *                                                                     *
238   *               Bits 3, 4, and 5 are decoded as follows (see also LCC.H): *
239   *                                                                     *
240   *               Bit  3  4  5   Function Return Value   Baud Rate      *
241   *               ------------------------------------------------      *
242   *                    0  0  0        0 (decimal)          300          *
243   *                    0  0  1        1 (decimal)          600          *
244   *                    0  1  0        2 (decimal)          1200         *
245   *                    0  1  1        3 (decimal)          2400         *
246   *                    1  0  0        4 (decimal)          4800         *
247   *                    1  0  1        5 (decimal)          9600         *
248   *                    1  1  0        6 (decimal)          19200        *
249   *                    1  1  1        7 (decimal)          38400 (57600) *
250   *                                                                     *
251   *               If a 14 MHz CPU is being used (I80152), then the max  *
252   *               baud rate is 38400, otherwise it is assumed that an   *
253   *               11 MHz CPU (or similar flavor) is being used and the max *
254   *               baud rate is 57600. So, the baud rate selected by code 111 *
255   *               is either 38400 or 57600 depending upon the target CPU. *
256   *                                                                     *
257   *               For the IBM-AT (8250):                                *
258   *                                                                     *
259   *               The baud rate is taken from global variable in MAIN.C. *
260   *                                                                     *
261   *               For all other processors:                             *
262   *                                                                     *
263   *               The baud rate defaults to 19200.                      *
264   *                                                                     *
265   * Input:        lcc_baud();                                           *
266   *                                                                     *
267   * Output:       Returns the baud rate code for external serial        *
268   *               communications as given by the table above.           *
269   *                                                                     *
270   * Globals:      lcd_error : module LCD.C                              *
271   *               8031 regs : module REG152.H                           *
272   *               lcc_baud_rate : module MAIN.C                         *
273   *                                                                     *
274   * Edit History: 08/14/90 - Written by Robin T. Laird.                 *
275   *                3/14/91 - Added code for IBM-At, Robin T. Laird.     *
276   *                                                                     *
277   \*********************************************************************/
278
279   static int lcc_baud()
280   {
281     int baud_rate;
282
283     lcd_error = AOK;                        /* Assume function successful...  */
284
285     /* Decode baud rate from bits 3, 4, and 5 of parallel port P1.        */
286
287     #if defined(I80152)
288     switch((byte)BRS2<<2 | (byte)BRS1<<1 | (byte)BRS0)
289     {
290       case 0:
291         baud_rate = BR300;
292         break;
```

```
293         case 1:
294           baud_rate = BR600;
295           break;
296         case 2:
297           baud_rate = BR1200;
298           break;
299         case 3:
300           baud_rate = BR2400;
301           break;
302         case 4:
303           baud_rate = BR4800;
304           break;
305         case 5:
306           baud_rate = BR9600;
307           break;
308         case 6:
309           baud_rate = BR19200;
310           break;
311         case 7:
312           baud_rate = MAX_BAUD_RATE;        /* Either 38.4K or 57.6K.        */
313           break;
314         default:
315           lcd_error = ERR_BAUD_RATE;
316           baud_rate = 0;
317       }
318       #elif defined(IBMAT)
319       baud_rate = lcc_baud_rate;
320       #else
321       baud_rate = BR19200;
322       #endif
323
324       return (baud_rate);
325     }
326
327
328   /*****************************************************************************\
329   *                                lcc_crc16                                  *
330   *****************************************************************************
331   *                                                                           *
332   * Function:     Generates a simple 16-bit checksum used as the CRC for      *
333   *               detecting communication errors. CRC is calculated as the    *
334   *               one's complement of the sum of the values of all the bytes  *
335   *               in the packet (from p[0] through p[length-1]).              *
336   *                                                                           *
337   * Input:        lcc_crc16(                                                  *
338   *                 lcd_packet p; packet for which checksum is to calculated. *
339   *                 word length;  length of packet (data portion only).       *
340   *                                                                           *
341   * Output:       Unsigned integer result of checksum used as CRC.            *
342   *                                                                           *
343   * Globals:      lcd_error : module LCD.C                                    *
344   *                                                                           *
345   * Edit History: 08/14/90 - Written by Robin T. Laird.                       *
346   *                                                                           *
347   \*****************************************************************************/
348
349   static word lcc_crc16(p, length)
350   lcd_packet p;
351   word length;
352   {
353     word i, sum;
354
355     lcd_error = AOK;                        /* Assume function successful...  */
356
357     /* Add up the data bytes.                                                  */
358     /* Complement and return as checksum CRC.                                  */
359
360     for (i = sum = 0; i < length; i++) sum += p[i];
361     return (~sum );
362   }
363
364
365   /*****************************************************************************\
```



```
366   *                                lcc_sys_init                               *
367   *****************************************************************************
368   *                                                                           *
369   * Function:     Initializes the Local Serial/Parallel Channel.              *
370   *               Sets the global operational mode variable, lcc_io_mode.     *
371   *               If SERIAL operation is selected the following is done:      *
372   *                                                                           *
373   *               TIMER 1 is used to generate the parameter baud rate.        *
374   *               PCON, TMOD, and SMOD are initialized appropriately.         *
375   *               TIMER 1 is enabled to start baud rate generation.           *
376   *               LCC receiver is enabled.                                    *
377   *                                                                           *
378   *               If PARALLEL operation is selected the following is done:    *
379   *                                                                           *
380   *               Baud rate parameter is ignored.                             *
381   *               Port direction register is set to input.                    *
382   *               Control registers are cleared.                              *
383   *               Data port is cleared.                                       *
384   *                                                                           *
385   * Input:        lcc_sys_init(                                               *
386   *                 byte mode;      I/O operational mode (serial/parallel).   *
387   *                 byte baud_rate; baud rate for serial operation.           *
388   *               );                                                          *
389   *                                                                           *
390   * Output:       Nothing.                                                    *
391   *                                                                           *
392   * Globals:      lcd_error   : module LCD.C                                  *
393   *               lcc_io_mode : module LCC.C                                  *
394   *               lcc_com_id  : module LCC.C                                  *
395   *               lcc_com_set : module LCC.C                                  *
396   *               8031 regs   : module REG152.H                               *
397   *               lcc_com_port    : module MAIN.C                             *
398   *               lcc_baud_rate   : module MAIN.C                             *
399   *               lcc_tty_control : module MAIN.C                             *
400   *               lcc_irq3_vect   : module MAIN.C                             *
401   *               lcc_irq4_vect   : module MAIN.C                             *
402   *               lcc_ic_ocw1     : module MAIN.C                             *
403   *                                                                           *
404   * Edit History: 08/14/90 - Written by Robin T. Laird.                       *
405   *               03/14/91 - Added code for IBM-At, Robin T. Laird.           *
406   *                                                                           *
407   \*****************************************************************************/
408
409   #if defined(IBMAT)
410
411   /* IBM-At 8250 and 8259 local definitions.                                  */
412   /* Interrupt numbers assigned to IRQs.                                      */
413
414   #define IRQ3                    0x0B    /* Interrupt request 3.           */
415   #define IRQ4                    0x0C    /* Interrupt request 4.           */
416
417   /* IRQs assigned to COM ports (default).                                    */
418   /* Note that COM1/COM3 share interrupts, and COM2/COM4 share interrupts.    */
419
420   #define COM1_INT_NUMBER         IRQ4
421   #define COM2_INT_NUMBER         IRQ3
422   #define COM3_INT_NUMBER         IRQ4
423   #define COM4_INT_NUMBER         IRQ3
424
425   #define MAX_RFIFO_READS         10              /* Clear this many bytes. */
426
427   #else
428
429   /* 8031 register values for initializing the local serial channel (LSC).   */
430   /* If using higher clock speed, SMOD is set to 0, otherwise set to 1.       */
431   /* SMOD is the MSB of the PCON Power Control register on the 8031.          */
432
433   #define SERIAL_PORT_MODE_1      0x50            /* SCON = 0bx1x1xxxx      */
434   #define SERIAL_PORT_MASK        0x5F            /* SCON = 0b0101xxxx      */
435   #define TIMER_1_MODE_2          0x20            /* TMOD = 0bxx1xxxxx      */
436   #define TIMER_1_MASK            0x2F            /* TMOD = 0b0010xxxx      */
437
438   #if defined(I80152)
```

```
439  #define DOUBLE_BAUD_RATE        0x00            /* PCON = 0b0xxxxxxx SMOD=0 */
440  #else
441  #define DOUBLE_BAUD_RATE        0x80            /* PCON = 0b1xxxxxxx SMOD=1 */
442  #endif
443
444  #endif
445
446  static void lcc_sys_init(mode, baud_rate)
447  int mode;
448  int baud_rate;
449  {
450    byte i, temp;
451
452    lcd_error = AOK;                        /* Assume function successful...    */
453
454    /* Initialize the I/O channel depending upon the mode...                      */
455
456    /* For serial mode operation:                                                 */
457    /*   For 8031 or 80152:                                                       */
458    /*     TIMER 1 is used in MODE 2 for variable baud rates.                     */
459    /*     TH1 sets TIMER 1 re-load value.                                        */
460    /*     SCON controls the serial port modes settings.                          */
461    /*     TMOD controls the timer mode settings.                                 */
462    /*     TCON controls the timer itself (turns it on/off).                      */
463    /*     SMOD controls baud rate doubling depending upon CPU speed.             */
464    /*     Clear serial buffer by reading data.                                   */
465    /*   For 8250 (IBM-AT) (NOSC serial port library functions are used):         */
466    /*     Configure 8250 with baud rate and serial params defined elsewhere.     */
467    /*     Disable serial interrupts for now.                                     */
468    /*     Clear all registers (MSR, LSR, and data buffer).                       */
469    /*     Save old interrupt vectors (so someone else can restore).              */
470    /*     Save old 8259 interrupt control word (so someone else can restore).    */
471    /*     Enable processor bus interrupts via the MODEM control register.        */
472
473    /* For parallel operation:                                                    */
474    /*   Set port direction to input.                                             */
475    /*   Clear data control lines.                                                */
476    /*   Clear data port.                                                         */
477
478    switch(mode)
479    {
480      case LCC_SIO_MODE:
481        lcc_io_mode = LCC_SIO_MODE;
482
483        #if defined(IBMAT)
484
485        /* Disable interrupts while we're changing vectors, ports, etc.         */
486
487        _disable();
488
489        /* Initialize 8250, see "Technical Reference Personal Computer AT".     */
490        /* Set baud rate (divisor latches).                                     */
491        /* DELAB must be 1 to write to baud rate divisor latches.               */
492        /* Parameter baud_rate is ignored.                                      */
493
494        outp(lcc_com_port+CC_LCR, CC_DLAB1);
495        outp(lcc_com_port+CC_DMSB, (lcc_baud_rate >> 8) & 0xFF);
496        outp(lcc_com_port+CC_DLSB, lcc_baud_rate & 0xFF);
497
498        /* Set serial port operational parameters.                              */
499        /* DELAB must be 0 to write/read LCR, IER, and data registers.          */
500
501        outp(lcc_com_port+CC_LCR, CC_DLAB0 | lcc_tty_control);
502
503        /* Disable all 8250 interrupts for now.                                 */
504
505        outp(lcc_com_port+CC_IER, CC_DISABLE_INT);
506
507        /* Clear line status and modem status registers.                        */
508
509        inp(lcc_com_port+CC_LSR);
510        inp(lcc_com_port+CC_MSR);
511
```



```
512        /* Clear chars from receive register.                                   */
513        /* Line status register (bit 0) indicates if data ready.                */
514
515        for (i = 0; i < MAX_RFIFO_READS; i++)
516        {
517         if (inp(lcc_com_port+CC_LSR) & CC_DATA_READY)
518           inp(lcc_com_port+CC_RBR);
519         else
520           break;
521        }
522        if (i > MAX_RFIFO_READS || inp(lcc_com_port+CC_LSR) & CC_DATA_READY)
523        {
524          lcd_error = ERR_RFIFO_NOT_CLEAR;
525          return;
526        }
527
528        /* Save old interrupt vectors so we can restore them later.             */
529        /* Set new vectors to our interrupt routines.                           */
530
531        switch(lcc_com_port)
532        {
533          case COM1:
534            lcc_irq4_vect = _dos_getvect(COM1_INT_NUMBER);
535            _dos_setvect(COM1_INT_NUMBER, lcc_sio1_int);
536            break;
537
538          case COM3:
539            lcc_irq4_vect = _dos_getvect(COM3_INT_NUMBER);
540            _dos_setvect(COM3_INT_NUMBER, lcc_sio3_int);
541            break;
542
543          case COM2:
544            lcc_irq3_vect = _dos_getvect(COM2_INT_NUMBER);
545            _dos_setvect(COM2_INT_NUMBER, lcc_sio2_int);
546            break;
547
548          case COM4:
549            lcc_irq3_vect = _dos_getvect(COM4_INT_NUMBER);
550            _dos_setvect(COM4_INT_NUMBER, lcc_sio4_int);
551            break;
552
553          default:
554            break;
555        }
556
557        /* Initialize the 8259 interrupt controller.                            */
558        /* Save old control interrupt mask.                                     */
559
560        lcc_ic_ocw1 = inp(IC_OCW1);
561
562        /* Set MODEM control reg to enable bus interrupts.                      */
563
564        outp(lcc_com_port+CC_MCR, CC_MODEM_BUS_ENABLE);
565
566        #else
567
568        /* Disable processor interrupts while we're changing things.            */
569
570        EA = 0;
571
572        PCON |= DOUBLE_BAUD_RATE;           /* Set Power Control Mode register. */
573        TH1   = (byte)baud_rate;            /* Set TIMER 1 auto-reload value.   */
574        TMOD |= TIMER_1_MODE_2;             /* Set Timer Mode Control register. */
575        TMOD &= TIMER_1_MASK;
576        SCON |= SERIAL_PORT_MODE_1;         /* Set Serial Port Control register.*/
577        SCON &= SERIAL_PORT_MASK;
578        TR1   = 1;                          /* Set TIMER 1 run bit.             */
579        temp  = SBUF;                       /* Clear serial receive buffer.     */
580
581        #endif
582        break;
583
584      case LCC_PIO_MODE:
```

```
585         lcc_io_mode = LCC_PIO_MODE;
586
587         #if defined(IBMAT)
588         lcd_error = ERR_IO_MODE;
589         #else
590
591         /* Disable processor interrupts while we're changing things.          */
592
593         PA = 0;
594
595         PDIR  = PIO_INPUT;                  /* Parallel I/O direction.        */
596         RTSO  = 0;                          /* Data control lines...          */
597         CTSI  = 0;
598         RTSI  = 0;
599         CTSO  = 0;
600         PIO   = 0;                          /* Parallel I/O data port.        */
601
602         #endif
603         break;
604
605       default:
606         lcd_error = ERR_IO_MODE;
607         return;
608     }
609   }
610
611
612   /*****************************************************************************\
613   *                              lcc_set_rcv_dst                              *
614   *****************************************************************************
615   *                                                                           *
616   * Function:     Sets the communications channel receiver destination address *
617   *               and packet size. The destination for the reception is the   *
618   *               parameter packet address. Special care must be taken so that *
619   *               data in the destination is not over written (or processed)  *
620   *               before the entire packet is received.                       *
621   *                                                                           *
622   * Input:        lcc_set_rcv_dst(                                            *
623   *                 lcd_packet p; pointer to receive packet (dest).           *
624   *                 word length;  size of receive packet (in bytes).          *
625   *               );                                                          *
626   *                                                                           *
627   * Output:       Nothing.                                                    *
628   *                                                                           *
629   * Globals:      lcd_error : module LCD.C                                    *
630   *                                                                           *
631   * Edit History: 03/09/91 - Written by Robin T. Laird.                       *
632   *                                                                           *
633   \*****************************************************************************/
634
635   static void lcc_set_rcv_dst(p, length)
636   lcd_packet p;
637   word length;
638   {
639     lcd_error = AOK;                        /* Assume function successful...  */
640   }
641
642
643   /*****************************************************************************\
644   *                              lcc_set_xmt_src                              *
645   *****************************************************************************
646   *                                                                           *
647   * Function:     Sets the communications channel transmission source address *
648   *               and packet size. The source for the transmission is the     *
649   *               parameter packet address. Special care must be taken so     *
650   *               that data in the source is not over written (or processed)  *
651   *               before the entire packet is transmitted.                    *
652   *                                                                           *
653   * Input:        lcc_set_xmt_src(                                            *
654   *                 lcd_packet p; pointer to transmit packet.                 *
655   *                 word length;  size of receive packet (in bytes).          *
656   *               );                                                          *
657   *                                                                           *
```



```
658   * Output:       Nothing.                                                    *
659   *                                                                           *
660   * Globals:      lcd_error : module LCD.C                                    *
661   *                                                                           *
662   * Edit History: 03/09/91 - Written by Robin T. Laird.                       *
663   *                                                                           *
664   \*****************************************************************************/
665
666   static void lcc_set_xmt_src(p, length)
667   lcd_packet p;
668   word length;
669   {
670     lcd_error = AOK;                        /* Assume function successful...  */
671   }
672
673
674   /*****************************************************************************\
675   *                              lcc_start_rcv                                *
676   *****************************************************************************
677   *                                                                           *
678   * Function:     Initiates reception of a data packet.                       *
679   *               Enables the LCC/LPC interrupt service routines.             *
680   *                                                                           *
681   * Input:        lcc_start_rcv();                                            *
682   *                                                                           *
683   * Output:       Nothing.                                                    *
684   *                                                                           *
685   * Globals:      lcd_error    : module LCD.C                                 *
686   *               lcc_io_mode  : module LCC.C                                 *
687   *               8031 regs    : module REG152.H                              *
688   *               lcc_com_port : module MAIN.C                                *
689   *                                                                           *
690   * Edit History: 07/10/90 - Written by Richard P. Smurlo and Robin T. Laird. *
691   *               03/14/91 - Added code for IBM-At, Robin T. Laird.           *
692   *                                                                           *
693   \*****************************************************************************/
694
695   static void lcc_start_rcv()
696   {
697     lcd_error = AOK;                        /* Assume function successful...  */
698
699     /* Enable processor interrupts according to operational mode.             */
700
701     #if defined(IBMAT)
702
703     /* Enable interrupts on COM port IRQ (0 a bit to enable IRQ int).         */
704     /* Assumes that COM1/COM3 on IRQ4 and COM2/COM4 on IRQ3).                 */
705
706     switch(lcc_com_port)
707     {
708       case COM1:
709       case COM3:
710         outp(IC_OCW1, inp(IC_OCW1) & IC_ENABLE_IRQ4);
711         break;
712
713       case COM2:
714       case COM4:
715         outp(IC_OCW1, inp(IC_OCW1) & IC_ENABLE_IRQ3);
716         break;
717
718       default:
719         break;
720     }
721
722     /* Enable receive and line status interrupts on the 8250.                 */
723
724     outp(lcc_com_port+CC_IER, inp(lcc_com_port+CC_IER) | CC_ENABLE_RCV
725                                                        | CC_ENABLE_STAT);
726
727     #else
728
729     switch(lcc_io_mode)
730     {
```

```
731       case LCC_SIO_MODE:
732         ES = 1;                             /* Enable serial interrupts.       */
733         break;
734
735       case LCC_PIO_MODE:
736         EX0 = 1;                            /* Enable external interrupt 0.    */
737         EX1 = 1;                            /* Enable external interrupt 1.    */
738         break;
739
740       default:
741         break;
742     }
743
744     #endif
745   }
746
747
748   /*****************************************************************************\
749   *                              lcc_start_xmt                                  *
750   *****************************************************************************
751   *                                                                           *
752   * Function:     Initiates transmission of data packets.                     *
753   *               Enables the LCC/LPC interrupt service routines.             *
754   *                                                                           *
755   * Input:        lcc_start_xmt();                                            *
756   *                                                                           *
757   * Output:       Nothing.                                                    *
758   *                                                                           *
759   * Globals:      lcd_error    : module LCD.C                                 *
760   *               lcc_io_mode  : module LCC.C                                 *
761   *               8031 regs    : module REG152.H                              *
762   *               lcc_com_port : module MAIN.C                                *
763   *                                                                           *
764   * Edit History: 07/10/90 - Written by Richard P. Smurlo and Robin T. Laird. *
765   *               03/14/91 - Added code for IBM-At, Robin T. Laird.           *
766   *                                                                           *
767   \*****************************************************************************/
768
769   static void lcc_start_xmt()
770   {
771     lcd_error = AOK;                          /* Assume function successful...   */
772
773     /* Enable processor interrupts according to operational mode.              */
774
775     /* If serial mode is used, have to cause interrupt to happen.              */
776     /* This is done by setting the transmit interrupt flag (TI).               */
777
778     #if defined(IBMAT)
779
780     /* Enable interrupts on COM port IRQ (0 a bit to enable IRQ int).          */
781     /* Assumes that COM1/COM3 on IRQ4 and COM2/COM4 on IRQ3).                  */
782
783     switch(lcc_com_port)
784     {
785       case COM1:
786       case COM3:
787         outp(IC_OCW1, inp(IC_OCW1) & IC_ENABLE_IRQ4);
788         break;
789
790       case COM2:
791       case COM4:
792         outp(IC_OCW1, inp(IC_OCW1) & IC_ENABLE_IRQ3);
793         break;
794
795       default:
796         break;
797     }
798
799     /* Enable transmit interrupts on the 8250.                                 */
800
801     outp(lcc_com_port+CC_IER, inp(lcc_com_port+CC_IER) | CC_ENABLE_XMT);
802
803     /* Call interrupt routine to start transmissions.                          */
```



```
804     /* Interrupt won't happen until interrupts are actually enabled.           */
805
806     switch(lcc_com_port)
807     {
808       case COM1:
809         lcc_sio1_int();
810         break;
811
812       case COM2:
813         lcc_sio2_int();
814         break;
815
816       case COM3:
817         lcc_sio3_int();
818         break;
819
820       case COM4:
821         lcc_sio4_int();
822         break;
823
824       default:
825         break;
826     }
827
828     #else
829
830     switch(lcc_io_mode)
831     {
832       case LCC_SIO_MODE:
833         ES = 1;                               /* Enable serial interrupts.       */
834         TI = 1;                               /* Cause interrupt to happen.      */
835         break;
836
837       case LCC_PIO_MODE:
838         EX0 = 1;                              /* Enable external interrupt 0.    */
839         EX1 = 1;                              /* Enable external interrupt 1.    */
840         break;
841
842       default:
843         break;
844     }
845
846     #endif
847   }
848
849
850   /*****************************************************************************\
851   *                              lcc_stop_rcv                                   *
852   *****************************************************************************
853   *                                                                           *
854   * Function:     Disables the LCC/LPC receiver by turning off the enable bit. *
855   *               Any incoming packet is dropped.                             *
856   *                                                                           *
857   * Input:        lcc_stop_rcv();                                             *
858   *                                                                           *
859   * Output:       Nothing.                                                    *
860   *                                                                           *
861   * Globals:      lcd_error    : module LCD.C                                 *
862   *               lcc_io_mode  : module LCC.C                                 *
863   *               8031 regs    : module REG152.H                              *
864   *               lcc_com_port : module MAIN.C                                *
865   *               lcc_ic_ocw1  : module MAIN.C                                *
866   *                                                                           *
867   * Edit History: 07/10/90 - Written by Richard P. Smurlo.                    *
868   *               03/14/91 - Added code for IBM-At, Robin T. Laird.           *
869   *                                                                           *
870   \*****************************************************************************/
871
872   static void lcc_stop_rcv()
873   {
874     lcd_error = AOK;                          /* Assume function successful...   */
875
876     /* Disable processor interrupts according to operational mode.             */
```

```
877
878     #if defined(IBMAT)
879
880     /* Disable receive and line status interrupts on the 8250.                */
881
882     outp(lcc_com_port+CC_IER, inp(lcc_com_port+CC_IER) & -CC_ENABLE_RCV
883                                                        & -CC_ENABLE_STAT);
884
885     /* Disable IRQ3 and IRQ4 interrupts on 8259.                              */
886     /* Restore old 8259 interrupt control bit mask.                           */
887
888     outp(IC_OCW1, lcc_ic_ocw1);
889
890     #else
891
892     switch(lcc_io_mode)
893     {
894       case LCC_SIO_MODE:
895         ES = 0;                          /* Disable serial interrupts.       */
896         break;
897
898       case LCC_PIO_MODE:
899         EX0 = 0;                         /* Disable external interrupt 0.    */
900         EX1 = 0;                         /* Disable external interrupt 1.    */
901         break;
902
903       default:
904         break;
905     }
906
907     #endif
908   }
909
910
911   /*************************************************************************\
912   *                              lcc_stop_xmt                               *
913   *************************************************************************
914   *                                                                       *
915   * Function:     Disables the LCC/LPC transmitter by turning off enable bit.  *
916   *               Any outgoing packet is terminated.                      *
917   *                                                                       *
918   * Input:        lcc_stop_xmt();                                         *
919   *                                                                       *
920   * Output:       Nothing.                                                *
921   *                                                                       *
922   * Globals:      lcd_error    : module LCD.C                             *
923   *               lcc_io_mode  : module LCC.C                             *
924   *               8031 regs    : module REG152.H                          *
925   *               lcc_com_port : module MAIN.C                            *
926   *               lcc_ic_ocw1  : module MAIN.C                            *
927   *                                                                       *
928   * Edit History: 07/10/90 - Written by Richard P. Smurlo.                *
929   *               03/14/91 - Added code for IBM-At, Robin T. Laird.       *
930   *                                                                       *
931   \*************************************************************************/
932
933   static void lcc_stop_xmt()
934   {
935     lcd_error = ACK;                     /* Assume function successful...  */
936
937     /* Disable processor interrupts according to operational mode.          */
938
939     #if defined(IBMAT)
940
941     /* Disable transmit interrupts on the 8250.                             */
942
943     outp(lcc_com_port+CC_IER, inp(lcc_com_port+CC_IER) & -CC_ENABLE_XMT);
944
945     /* Disable IRQ3 and IRQ4 interrupts on 8259.                            */
946     /* Restore old 8259 interrupt control bit mask.                         */
947
948     outp(IC_OCW1, lcc_ic_ocw1);
949
```



```
950     #else
951
952     switch(lcc_io_mode)
953     {
954       case LCC_SIO_MODE:
955         ES = 0;                          /* Disable serial interrupts.       */
956         break;
957
958       case LCC_PIO_MODE:
959         EX0 = 0;                         /* Disable external interrupt 0.    */
960         EX1 = 0;                         /* Disable external interrupt 1.    */
961         break;
962
963       default:
964         break;
965     }
966
967     #endif
968   }
969
970
971   /*************************************************************************\
972   *                           lcc_enable_interrupts                       *
973   *************************************************************************
974   *                                                                       *
975   * Function:     Enables processor interrupts for the 8031/80152.        *
976   *                                                                       *
977   * Input:        lcc_enable_interrupts();                                *
978   *                                                                       *
979   * Output:       Nothing.                                                *
980   *                                                                       *
981   * Globals:      lcd_error    : module LCD.C                             *
982   *               8031 regs    : module REG152.H                          *
983   *               lcc_com_port : module MAIN.C                            *
984   *                                                                       *
985   * Edit History: 03/09/91 - Written by Richard P. Smurlo.                *
986   *               03/14/91 - Added code for IBM-At, Robin T. Laird.       *
987   *                                                                       *
988   \*************************************************************************/
989
990   static void lcc_enable_interrupts()
991   {
992     lcd_error = AOK;                     /* Assume function successful...  */
993
994     #if defined(IBMAT)
995     _enable();                           /* Enable all interrupts (STI).   */
996     #else
997     EA = 1;                              /* Enable all interrupts.         */
998     #endif
999   }
1000
1001
1002  /*************************************************************************\
1003  *                                lcc_sio_int                            *
1004  *************************************************************************
1005  *                                                                       *
1006  * Function:     Processes a valid serial interrupt (ESV) from the LSC.  *
1007  *               This routine is called upon completion of a serial interrupt.*
1008  *               This happens after a char has been transmitted or after a *
1009  *               char has been received. The routine determines which int  *
1010  *               has occured and proceeds accordingly.                   *
1011  *                                                                       *
1012  *               This interrupt routine services the global rcv_buffer.  *
1013  *               As bytes are received, they are placed in the next position *
1014  *               in the receive buffer, and the rear buffer index is updated. *
1015  *               If the buffer is full, then bytes are dropped. The full flag *
1016  *               is also updated.                                        *
1017  *                                                                       *
1018  *               This interrupt routine also services the global xmt_buffer. *
1019  *               As bytes are transmitted, they are removed from the next  *
1020  *               position in the transmit buffer, and the front buffer index *
1021  *               is updated. If the buffer is empty, then transmission is  *
1022  *               terminated. The full empty is also updated.             *
```

```
1023  *                                                                       *
1024  * Input:        lcc_sio_int() interrupt;      I80152 and I8031          *
1025  *               lcc_sio1_int() interrupt;     IBMAT COM1                *
1026  *               lcc_sio2_int() interrupt;     IBMAT COM2                *
1027  *               lcc_sio3_int() interrupt;     IBMAT COM3                *
1028  *               lcc_sio4_int() interrupt;     IBMAT COM4                *
1029  *                                                                       *
1030  * Output:       Nothing.                                                *
1031  *                                                                       *
1032  * Globals:      rcv_buffer : module BMF.C                               *
1033  *               xmt_buffer : module BMF.C                               *
1034  *                lcd_state : module LCD.C                               *
1035  *               8031_regs  : module REG152.H                            *
1036  *                                                                       *
1037  * Edit History: 07/10/90 - Written by Robin T. Laird.                   *
1038  *               03/14/91 - Added code for IBM-AT, Robin T. Laird.       *
1039  *                                                                       *
1040  \***********************************************************************/
1041
1042  #if defined(IBMAT)
1043
1044  unsigned int ctr = 0;
1045  unsigned int errctr = 0;
1046
1047  static void interrupt far lcc_sio1_int()
1048  {
1049    static byte b;
1050
1051    /* Interrupt occured on COM1.                                         */
1052    /* When LSB of Interrupt ID register set then no more interrupts pending. */
1053    /* Loop until this condition is met. This allows multiple int servicing. */
1054
1055    do {
1056      /* Increment interrupt service counter.                             */
1057
1058      ++ctr;
1059
1060      /* Interrupt ID register holds number of interrupt source(s).       */
1061      /* Transmit interrupt is serviced last to avoid interrupting ourself. */
1062
1063      switch(inp(COM1+CC_IIR))
1064      {
1065        case MODEM_INT:
1066        {
1067          /* Should never have one of these.                              */
1068          /* Just clear line status and modem status registers.           */
1069
1070          /* Reading the MSR register clears int in IIR.                  */
1071
1072          inp(COM1+CC_LSR);
1073          inp(COM1+CC_MSR);
1074        }
1075        break;
1076
1077        case LINE_STATUS_ERR_INT:
1078        {
1079          /* Get (and clear) line status.                                 */
1080          /* Check status to see if there is a byte waiting to be received. */
1081          /* If there is, get and throw away bad byte.                    */
1082          /* Otherwise, all done.                                         */
1083
1084          /* Reading the line status register clears int in IIR.          */
1085
1086          if (inp(COM1+CC_LSR) & CC_DATA_READY)
1087          {
1088            inp(COM1+CC_RBR);
1089          ++errctr;
1090          }
1091        }
1092        break;
1093
1094        case RCV_DATA_AVAIL_INT:
1095        {
```



```
1096          /* Get the byte from the receive buffer (put it in b).        */
1097          /* Check for receive errors in LSR (if any err bit set then error). */
1098          /* If bad byte or buffer full throw byte away.                */
1099          /* Otherwise...                                               */
1100          /*   Put byte at end of buffer.                               */
1101          /*   Increment rear pointer.                                  */
1102          /*   Check if buffer full, and set flag appropriately.        */
1103
1104          /* Reading the receive buffer register clears int in IIR.     */
1105
1106          b = inp(COM1+CC_RBR);
1107          if ((inp(COM1+CC_LSR) & CC_RECEIVE_ERRS) || rcv_buffer.full)
1108          {
1109          }
1110          else
1111          {
1112            rcv_buffer.item[rcv_buffer.rear] = b;
1113            buf_inc(rcv_buffer.rear);
1114            if (rcv_buffer.front == rcv_buffer.rear) rcv_buffer.full = TRUE;
1115          }
1116        }
1117        break;
1118
1119        case XMT_EMPTY_INT:
1120        default:
1121        {
1122          /* Get (and clear) line status.                               */
1123          /* Make sure transmitter holding register is empty.           */
1124          /* If it's not, we got problems, so just return.              */
1125          /* If the transmit buffer is empty, just return also.         */
1126          /* Otherwise...                                               */
1127          /*   Output byte at front of buffer.                          */
1128          /*   Increment front pointer.                                 */
1129          /*   Buffer can't be full since we just transmitted a byte.   */
1130
1131          /* Reading IIR or writing to transmit buff reg clears int in IIR. */
1132
1133          if ((inp(COM1+CC_LSR) & CC_TRANSMIT_HOLD_EMPTY) && !xmt_buffer.empty)
1134          {
1135            xmt_buffer.full = FALSE;
1136            outp(COM1+CC_THR, xmt_buffer.item[xmt_buffer.front]);
1137            buf_inc(xmt_buffer.front);
1138            if (xmt_buffer.front == xmt_buffer.rear) xmt_buffer.empty = TRUE;
1139          }
1140          else
1141          {
1142          }
1143        }
1144      }
1145    } while (inp(COM1+CC_IIR) != 1);
1146
1147    /* Acknowledge interrupt on 8259 by sending end-of-interrupt command. */
1148
1149    outp(IC_OCW2, IC_EOI);
1150  }
1151
1152  static void interrupt far lcc_sio2_int()
1153  {
1154    static byte b;
1155
1156    /* Interrupt occured on COM2.                                         */
1157
1158    do {
1159      ++ctr;
1160      switch(inp(COM2+CC_IIR))
1161      {
1162        case MODEM_INT:
1163        {
1164          inp(COM2+CC_LSR);
1165          inp(COM2+CC_MSR);
1166        }
1167        break;
1168
```

```
1169        case LINE_STATUS_ERR_INT:
1170        {
1171          if (inp(COM2+CC_LSR) & CC_DATA_READY)
1172          {
1173            inp(COM2+CC_RBR);
1174          ++errctr;
1175          }
1176        }
1177        break;
1178
1179        case RCV_DATA_AVAIL_INT:
1180        {
1181          b = inp(COM2+CC_RBR);
1182          if ((inp(COM2+CC_LSR) & CC_RECEIVE_ERRS) || rcv_buffer.full)
1183          {
1184          }
1185          else
1186          {
1187            rcv_buffer.item[rcv_buffer.rear] = b;
1188            buf_inc(rcv_buffer.rear);
1189            if (rcv_buffer.front == rcv_buffer.rear) rcv_buffer.full = TRUE;
1190          }
1191        }
1192        break;
1193
1194        case XMT_EMPTY_INT:
1195        default:
1196        {
1197          if ((inp(COM2+CC_LSR) & CC_TRANSMIT_HOLD_EMPTY) && !xmt_buffer.empty)
1198          {
1199            xmt_buffer.full = FALSE;
1200            outp(COM2+CC_THR, xmt_buffer.item[xmt_buffer.front]);
1201            buf_inc(xmt_buffer.front);
1202            if (xmt_buffer.front == xmt_buffer.rear) xmt_buffer.empty = TRUE;
1203          }
1204          else
1205          {
1206          }
1207        }
1208      }
1209    } while (inp(COM2+CC_IIR) != 1);
1210    outp(IC_OCW2, IC_EOI);
1211  }
1212
1213  static void interrupt far lcc_sio3_int()
1214  {
1215    static byte b;
1216
1217    /* Interrupt occured on COM3.                                    */
1218
1219    do {
1220      ++ctr;
1221      switch(inp(COM3+CC_IIR))
1222      {
1223        case MODEM_INT:
1224        {
1225          inp(COM3+CC_LSR);
1226          inp(COM3+CC_MSR);
1227        }
1228        break;
1229
1230        case LINE_STATUS_ERR_INT:
1231        {
1232          if (inp(COM3+CC_LSR) & CC_DATA_READY)
1233          {
1234            inp(COM3+CC_RBR);
1235          ++errctr;
1236          }
1237        }
1238        break;
1239
1240        case RCV_DATA_AVAIL_INT:
1241        {
```



```
1242          b = inp(COM3+CC_RBR);
1243          if ((inp(COM3+CC_LSR) & CC_RECEIVE_ERRS) || rcv_buffer.full)
1244          {
1245          }
1246          else
1247          {
1248            rcv_buffer.item[rcv_buffer.rear] = b;
1249            buf_inc(rcv_buffer.rear);
1250            if (rcv_buffer.front == rcv_buffer.rear) rcv_buffer.full = TRUE;
1251          }
1252        }
1253        break;
1254
1255        case XMT_EMPTY_INT:
1256        default:
1257        {
1258          if ((inp(COM3+CC_LSR) & CC_TRANSMIT_HOLD_EMPTY) && !xmt_buffer.empty)
1259          {
1260            xmt_buffer.full = FALSE;
1261            outp(COM3+CC_THR, xmt_buffer.item[xmt_buffer.front]);
1262            buf_inc(xmt_buffer.front);
1263            if (xmt_buffer.front == xmt_buffer.rear) xmt_buffer.empty = TRUE;
1264          }
1265          else
1266          {
1267          }
1268        }
1269      }
1270    } while (inp(COM3+CC_IIR) != 1);
1271    outp(IC_OCW2, IC_EOI);
1272  }
1273
1274  static void interrupt far lcc_sio4_int()
1275  {
1276    static byte b;
1277
1278    /* Interrupt occured on COM4.                                    */
1279
1280    do {
1281      ++ctr;
1282      switch(inp(COM4+CC_IIR))
1283      {
1284        case MODEM_INT:
1285        {
1286          inp(COM4+CC_LSR);
1287          inp(COM4+CC_MSR);
1288        }
1289        break;
1290
1291        case LINE_STATUS_ERR_INT:
1292        {
1293          if (inp(COM4+CC_LSR) & CC_DATA_READY)
1294          {
1295            inp(COM4+CC_RBR);
1296          ++errctr;
1297          }
1298        }
1299        break;
1300
1301        case RCV_DATA_AVAIL_INT:
1302        {
1303          b = inp(COM4+CC_RBR);
1304          if ((inp(COM4+CC_LSR) & CC_RECEIVE_ERRS) || rcv_buffer.full)
1305          {
1306          }
1307          else
1308          {
1309            rcv_buffer.item[rcv_buffer.rear] = b;
1310            buf_inc(rcv_buffer.rear);
1311            if (rcv_buffer.front == rcv_buffer.rear) rcv_buffer.full = TRUE;
1312          }
1313        }
1314        break;
```

```
1315
1316        case XMT_EMPTY_INT:
1317        default:
1318        {
1319          if ((inp(COM4+CC_LSR) & CC_TRANSMIT_HOLD_EMPTY) && !xmt_buffer.empty)
1320          {
1321            xmt_buffer.full = FALSE;
1322            outp(COM4+CC_THR, xmt_buffer.item[xmt_buffer.front]);
1323            buf_inc(xmt_buffer.front);
1324            if (xmt_buffer.front == xmt_buffer.rear) xmt_buffer.empty = TRUE;
1325          }
1326          else
1327          {
1328          }
1329        }
1330      }
1331    } while (inp(COM4+CC_IIR) != 1);
1332    outp(IC_OCW2, IC_EOI);
1333  }
1334
1335  #else
1336
1337  static void lcc_sio_int() interrupt 4 using 2
1338  {
1339    /* Determine which interrupt occured (receive or transmit).              */
1340
1341    if (RI)
1342    {
1343      /* For a valid reception:                                              */
1344      /*   Clear receive interrupt flag.                                     */
1345      /*   Make sure buffer is not full (if it is, reset RI and drop chars). */
1346      /*   Move byte from input register to receive buffer.                  */
1347      /*   Adjust buffer rear index.                                         */
1348      /*   See if the buffer is full.                                        */
1349
1350      RI = 0;
1351      if (!rcv_buffer.full)
1352      {
1353        rcv_buffer.item[rcv_buffer.rear] = SBUF;
1354        buf_inc(rcv_buffer.rear);
1355        if (rcv_buffer.front == rcv_buffer.rear) rcv_buffer.full = TRUE;
1356      }
1357    }
1358
1359    if (TI)
1360    {
1361      /* For valid transmit interrupt:                                       */
1362      /*   Clear transmit interrupt flag.                                    */
1363      /*   Mark buffer as not full since we just removed an item.            */
1364      /*   Make sure buffer is not empty (if so just clear int flag).        */
1365      /*   Move byte from front of buffer to transmit register.              */
1366      /*   Adjust buffer front index.                                        */
1367      /*   See if the buffer is empty.                                       */
1368
1369      TI = 0;
1370      if (!xmt_buffer.empty)
1371      {
1372        xmt_buffer.full = FALSE;
1373        SBUF = xmt_buffer.item[xmt_buffer.front];
1374        buf_inc(xmt_buffer.front);
1375        if (xmt_buffer.front == xmt_buffer.rear) xmt_buffer.empty = TRUE;
1376      }
1377    }
1378  }
1379
1380  #endif
1381
1382
1383  /*****************************************************************************\
1384  *                                lcc_pio_int                                  *
1385  *****************************************************************************
1386  *                                                                           *
1387  * Function:     Processes valid parallel I/O interrupts.                    *
```



```
1388  *               Both parallel input and output requests are processed via   *
1389  *               this interrupt handler.                                     *
1390  *                                                                           *
1391  * Input:        lcc_pio_int() interrupt;                                    *
1392  *                                                                           *
1393  * Output:       Nothing.                                                    *
1394  *                                                                           *
1395  * Globals:      None.                                                       *
1396  *                                                                           *
1397  * Edit History: 07/10/90 - Written by Richard P. Smurlo.                    *
1398  *                                                                           *
1399  \*****************************************************************************/
1400
1401  #if defined(I80152)
1402
1403  static void lcc_pio_int() interrupt 0 using 2
1404  {
1405  }
1406
1407  #endif
```

```
1   /*************************************************************************\
2   *                                LCD.H                                    *
3   *************************************************************************
4   *                                                                         *
5   * CPCI:              MRA-COM-LCS-LCD-H-R0C0                               *
6   *                                                                         *
7   * Description:  LCS communications device handler variables and functions.*
8   *               Contains constant function parameter declarations as well *
9   *               as function return values (for success and failure of all *
10  *               operations). Contains the function prototypes for the LCD.C*
11  *               module.                                                   *
12  *                                                                         *
13  *               Module LCD exports the following types/variables/functions:*
14  *                                                                         *
15  *               typedef lcd_packet;                                       *
16  *               typedef lcd_state;                                        *
17  *                                                                         *
18  *               int lcd_error;                                            *
19  *                                                                         *
20  *               lcd_init();                                               *
21  *               lcd_reset();                                              *
22  *               lcd_enable();                                             *
23  *               lcd_disable();                                            *
24  *               lcd_receive_packet();                                     *
25  *               lcd_transmit_packet();                                    *
26  *               lcd_status();                                             *
27  *                                                                         *
28  * Notes:        1) See SDS pp. 5-6 through 5-x for more information.      *
29  *                                                                         *
30  * Edit History: 08/14/90 - Written by Robin T. Laird.                     *
31  *                                                                         *
32  \*************************************************************************/
33
34  /* Public Data Structures:                                               */
35
36  #ifndef LCD_MODULE_CODE
37  #define LCD_MODULE_CODE          3000
38
39  #define LCD_ERR_NOT_INIT         1+LCD_MODULE_CODE
40  #define LCD_ERR_PACKET_LENGTH    2+LCD_MODULE_CODE
41  #define LCD_ERR_NUM_ATTEMPTS     3+LCD_MODULE_CODE
42
43  #define LCD_ERR_RECEIVE_PACKET   4+LCD_MODULE_CODE
44  #define LCD_ERR_TRANSMIT_PACKET  5+LCD_MODULE_CODE
45
46  #define LCD_FAIL_RECEIVER        6+LCD_MODULE_CODE
47  #define LCD_FAIL_TRANSMITTER     7+LCD_MODULE_CODE
48
49  #define LCD_WAIT_FOREVER         65535
50  #define LCD_DONT_WAIT            0
51
52  #define LCD_MAX_PACKET_LENGTH    SYS_MAX_PACKET_SIZE
53  #define LCD_MAX_ATTEMPTS         60000
54
55  #define LCD_DEST_ADDR_POS        0
56  #define LCD_LEN_POS              1
57  #define LCD_SRC_ADDR_POS         2
58
59  /* Type for receive/transmit data packet, simply a 256-element array.    */
60
61  typedef byte lcd_packet[LCD_MAX_PACKET_LENGTH];
62
63  /* Structure type for LCD module status (holds rcv/xmt error counts).    */
64
65  typedef struct { word r_valid_cnt;
66                   word r_err_cnt;
67                   word r_crce_err_cnt;
68                   word r_bop_err_cnt;
69                   word x_valid_cnt;
70                   word x_err_cnt;
71                 } lcd_state;
72
73  /* External module global error variable.                               */
```



```
74
75  extern int lcd_error;
76
77  /* Public Functions:                                                     */
78
79  void lcd_init(void);
80  void lcd_reset(void);
81  void lcd_enable(void);
82  void lcd_disable(void);
83  void lcd_receive_packet(lcd_packet p, word retry);
84  void lcd_transmit_packet(lcd_packet p, word retry);
85  void lcd_status(lcd_state *s);
86
87  #endif
```

```
1   /*****************************************************************************\
2   *                                  LCD.C                                      *
3   *****************************************************************************
4   *                                                                            *
5   * CPCI:          JED90-MRA-COM-LCS-LCD-C-R0C0                                *
6   *                                                                            *
7   * Description:   LCS communications device handler functions.                *
8   *                Implements the MRA standard Local Communications            *
9   *                Device (LCD) Handler module. This module contains the       *
10  *                standard device handler functions and must include the      *
11  *                low-level data link layer functions for the actual hardware *
12  *                implementation (currently implemented for the 8031 LSC and  *
13  *                the 80C152 LSC/LPC - Local Serial/Parallel Channel).        *
14  *                                                                            *
15  *                Message format at this level (ISO OSI data link layer) is:  *
16  *                                                                            *
17  *                 | byte 0 | byte 1 | byte 2 | byte 3 | byte 4 | byte n|      *
18  *                 |-------------------------------------------------|        *
19  *                 |  DEST  | LENGTH | SOURCE |  xxxx  |  xxxx  | .... |      *
20  *                                                                            *
21  *                Module LCD exports the following variables/functions:       *
22  *                                                                            *
23  *                int lcd_error;                                              *
24  *                                                                            *
25  *                lcd_init();                                                 *
26  *                lcd_reset();                                                *
27  *                lcd_enable();                                               *
28  *                lcd_disable();                                              *
29  *                lcd_receive_packet();                                       *
30  *                lcd_transmit_packet();                                      *
31  *                lcd_status();                                               *
32  *                                                                            *
33  * Notes:         1) The files BMF.H and BMF.C contain the support functions  *
34  *                   for managing the receive and transmit circular buffers.  *
35  *                2) The files LCC.H and LCC.C contain the required support   *
36  *                   functions for the Local Communications Channel hardware. *
37  *                                                                            *
38  * Edit History: 08/14/90 - Written by Richard P. Smurlo and Robin T. Laird.  *
39  *                                                                            *
40  \*****************************************************************************/
41
42  #include <sysdefs.h>                    /* System constants and types.      */
43  #include "lcd.h"                        /* LCD public literals/functions.   */
44  #include "lcc.h"                        /* Local Communications Channel.    */
45  #include "bmf.h"                        /* Buffer management functions.     */
46
47  /* Public Variables:                                                        */
48
49  /* Global module error variable, lcd_error.                                 */
50  /* lcd_error contains code of last error occurence.                         */
51  /* Should be set to AOK after each successful function call.                */
52  /* Variable can be examined by other software after each function call.     */
53
54  XDATA int lcd_error = LCD_ERR_NOT_INIT; /* Global module error variable.    */
55
56  /* Global module state variable, _lcd_state.                                */
57  /* Holds LCD module error counts for packet reception/transmission.         */
58
59  static XDATA lcd_state _lcd_state;      /* Global module state variable.    */
60
61  /* Buffer management functions should be included here.                     */
62
63  #include "bmf.c"                        /* Buffer management functions.     */
64
65  /* Low-level data link layer support functions should be included here.     */
66
67  #include "lcc.c"                        /* Local communications channel.    */
68
69
70  /*****************************************************************************\
71  *                                 lcd_init                                    *
72  *****************************************************************************
73  *                                                                            *
```



```
74  * Function:      Initializes the Local Communications Device Handler.        *
75  *                Clears the transmit and receive buffers, obtains the local  *
76  *                communications channel I/O mode and baud rate, initializes  *
77  *                the Local Serial/Parallel Channel, and starts operation     *
78  *                of the LCC. The LCC reception/transmission error and valid  *
79  *                packet counters of the module state variable are cleared.   *
80  *                                                                            *
81  * Input:         lcd_init();                                                 *
82  *                                                                            *
83  * Output:        Nothing.                                                    *
84  *                                                                            *
85  * Globals:       lcd_error  : module LCD.C                                   *
86  *                _lcd_state : module LCD.C                                   *
87  *                rcv_buffer : module BMF.C                                   *
88  *                xmt_buffer : module BMF.C                                   *
89  *                                                                            *
90  * Edit History: 07/08/90 - Written by Robin T. Laird.                        *
91  *                                                                            *
92  \*****************************************************************************/
93
94  void lcd_init()
95  {
96    lcd_error = AOK;                      /* Assume function successful...    */
97
98    /* Reset all error counting registers in module state variable.           */
99    /* Valid reception/transmission counters are also cleared.                */
100
101   _lcd_state.r_valid_cnt     = 0;
102   _lcd_state.r_err_cnt       = 0;
103   _lcd_state.r_crce_err_cnt  = 0;
104   _lcd_state.r_bop_err_cnt   = 0;
105   _lcd_state.x_valid_cnt     = 0;
106   _lcd_state.x_err_cnt       = 0;
107
108   /* Initialize the transmit and receive buffers.                           */
109
110   buf_clear(&xmt_buffer);
111   buf_clear(&rcv_buffer);
112
113   /* Get the I/O mode and baud rate (if applicable) for this system.        */
114   /* Pass to LCC initialization function (which sets up I/O channel).       */
115   /* No errors are currently fatal (they are only warnings).                */
116
117   lcc_sys_init(lcc_mode(), lcc_baud());
118
119   /* Start the LCC receiver (begin receiving data packets).                 */
120   /* Start the LCC transmitter.                                             */
121   /* Errors are non-fatal and would cause only degraded performance.        */
122
123   lcc_start_rcv();
124   lcc_start_xmt();
125
126   /* Enable interrupts...                                                   */
127
128   lcc_enable_interrupts();
129 }
130
131
132 /*****************************************************************************\
133 *                                 lcd_reset                                   *
134 *****************************************************************************
135 *                                                                            *
136 * Function:      Performs a soft reset of the LCD systems.                   *
137 *                The receive and transmit buffers are cleared and the        *
138 *                receive and transmit destination and source addresses       *
139 *                for the LCC I/O channels are reset to the beginning of      *
140 *                the buffers. The LCC reception/transmission error and       *
141 *                valid packet counters of the module state structure are     *
142 *                cleared.                                                    *
143 *                                                                            *
144 * Input:         lcd_reset();                                                *
145 *                                                                            *
146 * Output:        Nothing.                                                    *
```

```
147   *                                                                       *
148   * Globals:      lcd_error  : module LCD.C                               *
149   *               _lcd_state : module LCD.C                               *
150   *               rcv_buffer : module BMF.C                               *
151   *               xmt_buffer : module BMF.C                               *
152   *                                                                       *
153   * Edit History: 07/09/90 - Written by Robin T. Laird.                   *
154   *                                                                       *
155   \***** ***************************************************************/
156
157   void lcd_reset()
158   {
159     lcd_error = AOK;                        /* Assume function successful...    */
160
161     /* Reset all error counting registers in module state variable.          */
162     /* Valid reception/transmission counters are also cleared.               */
163
164     _lcd_state.r_valid_cnt     = 0;
165     _lcd_state.r_err_cnt       = 0;
166     _lcd_state.r_crce_err_cnt  = 0;
167     _lcd_state.r_bop_err_cnt   = 0;
168     _lcd_state.x_valid_cnt     = 0;
169     _lcd_state.x_err_cnt       = 0;
170
171     /* Re-initialize the transmit and receive buffers.                       */
172
173     buf_clear(&xmt_buffer);
174     buf_clear(&rcv_buffer);
175   }
176
177
178   /*************************************************************************\
179   *                              lcd_enable                               *
180   *************************************************************************
181   *                                                                       *
182   * Function:     Enables reception and transmission of data packets.     *
183   *               The LCC receiver and transmitter are re-enabled.        *
184   *               The function assumes that the LCC has been disabled     *
185   *               for some reason. It is normally not necessary to enable *
186   *               the LCC after it has been initialized (by lcd_init()).  *
187   *                                                                       *
188   * Input:        lcd_enable();                                           *
189   *                                                                       *
190   * Output:       Nothing.                                                *
191   *                                                                       *
192   * Globals:      lcd_error : module LCD.C                                *
193   *                                                                       *
194   * Edit History: 07/09/90 - Written by Robin T. Laird.                   *
195   *                                                                       *
196   \*************************************************************************/
197
198   void lcd_enable()
199   {
200     lcd_error = AOK;                        /* Assume function successful...    */
201
202     /* Start both the receiver and transmitter.                              */
203
204     lcc_start_rcv();
205     lcc_start_xmt();
206   }
207
208
209   /*************************************************************************\
210   *                              lcd_disable                              *
211   *************************************************************************
212   *                                                                       *
213   * Function:     Disables reception and transmission of data packets.    *
214   *               The LCC receiver and transmitter are disabled.          *
215   *               Any incoming data will be lost (dropped). The LCC must  *
216   *               be re-enabled using lcd_enable() before packets may be  *
217   *               received or transmitted again.                          *
218   *                                                                       *
219   * Input:        lcd_disable();                                          *
```



```
220   *                                                                       *
221   * Output:       Nothing.                                                *
222   *                                                                       *
223   * Globals:      lcd_error : module LCD.C                                *
224   *                                                                       *
225   * Edit History: 07/09/90 - Written by Robin T. Laird.                   *
226   *                                                                       *
227   \*************************************************************************/
228
229   void lcd_disable()
230   {
231     lcd_error = AOK;                        /* Assume function successful...    */
232
233     /* Stop both the receiver and transmitter.                               */
234
235     lcc_stop_rcv();
236     lcc_stop_xmt();
237   }
238
239
240   /*************************************************************************\
241   *                           lcd_receive_packet                          *
242   *************************************************************************
243   *                                                                       *
244   * Function:     Removes a packet from the receive buffer if possible.   *
245   *               If a packet is available, it is removed from the receive*
246   *               buffer, and the packet length is updated. If a packet is not *
247   *               available from the receive buffer, the function will perform *
248   *               as follows depending upon the retry value:              *
249   *                                                                       *
250   *               LCD_DONT_WAIT    : remove packet if available, return if not.*
251   *               LCD_WAIT_FOREVER : wait forever for packet to be received.  *
252   *               retry            : try retry times to receive packet.   *
253   *                                                                       *
254   * Input:        lcd_receive_packet(                                     *
255   *                 lcd_packet p; packet to be received (destination).    *
256   *                 word retry;   number of times to try receiving packet.*
257   *               );                                                      *
258   *                                                                       *
259   * Output:       Nothing.                                                *
260   *                                                                       *
261   * Globals:      lcd_error  : module LCD.C                               *
262   *               rcv_buffer : module BMF.C                               *
263   *                                                                       *
264   * Edit History: 07/08/90 - Written by Robin T. Laird.                   *
265   *                                                                       *
266   \*************************************************************************/
267
268   void lcd_receive_packet(p, retry)
269   lcd_packet p;
270   word retry;
271   {
272     lcd_error = AOK;                        /* Assume function successful...    */
273
274     /* If we don't want to wait (LCD_DONT_WAIT):                             */
275     /*   Try and remove packet from buffer.                                  */
276     /*   If one not available then lcd_error = ERR_BUF_EMPTY.                */
277     /*   Else, we've successfully removed a received packet.                 */
278
279     /* If we want to wait forever (LCD_WAIT_FOREVER):                        */
280     /*   Wait until there is a packet in the buffer, then remove it.         */
281     /*   This function WILL wait forever, i.e., try indefinitely.            */
282
283     /* If we want to wait for a period of time:                              */
284     /*   Set up and zero auxilliary timer.                                   */
285     /*   Loop, checking to see if a packet is available and timer not expired.*/
286     /*   If the timer expires, indicate buffer empty and return.             */
287     /*   Else, get incoming packet and return.                               */
288
289     if (retry == LCD_DONT_WAIT)
290     {
291       buf_remove(&rcv_buffer, p);
292     }
```

```
293     else if (retry == LCD_WAIT_FOREVER)
294     {
295       while (buf_empty(&rcv_buffer));
296       buf_remove(&rcv_buffer, p);
297     }
298     else if (retry <= LCD_MAX_ATTEMPTS)
299     {
300       while (buf_empty(&rcv_buffer))
301         if (retry-- == 0)
302         {
303           lcd_error = LCD_ERR_RECEIVE_PACKET;
304           return;
305         }
306       buf_remove(&rcv_buffer, p);
307     }
308     else
309     {
310       lcd_error   LCD_ERR_NUM_ATTEMPTS;
311     }
312
313     /* If an error occured while removing packet from buffer, log type.      */
314     /* If no error ocurred, log successful reception.                         */
315
316     if (lcd_error == ERR_BOP_NOT_FOUND)
317     {
318       _lcd_state.r_bop_err_cnt++;
319       _lcd_state.r_err_cnt++;
320     }
321     else if (lcd_error == ERR_CRC_INVALID)
322     {
323       _lcd_state.r_crce_err_cnt++;
324       _lcd_state.r_err_cnt++;
325     }
326     else if (lcd_error == AOK)
327       _lcd_state.r_valid_cnt++;
328   }
329
330
331   /*************************************************************************\
332   *                          lcd_transmit_packet                          *
333   *************************************************************************
334   *                                                                       *
335   * Function:     Adds the parameter packet to the transmit buffer if possible.*
336   *               If the transmit buffer is full an error is generated.   *
337   *               Otherwise, if the buffer is empty, the packet is transmitted *
338   *               immediately, and the packet information is inserted into the *
339   *               transmit buffer. Depending upon the retry value, the    *
340   *               function will perform as follows:                       *
341   *                                                                       *
342   *               LCD_DONT_WAIT    : return immediately, pkt added to buffer.  *
343   *               LCD_WAIT_FOREVER : wait forever for packet to be transmitted.*
344   *               retry            : try retry times to transmit packet.  *
345   *                                                                       *
346   *               Currently, only the LCD_WAIT_FOREVER option is supported.   *
347   *               This is equivalent to a one packet deep transmit buffer.    *
348   *                                                                       *
349   * Input:        lcd_transmit_packet(                                    *
350   *                 lcd_packet p; packet to be transmitted.               *
351   *                 word retry;   number of times to try transmitting packet.  *
352   *               );                                                      *
353   *                                                                       *
354   * Output:       Nothing.                                                *
355   *                                                                       *
356   * Globals:      lcd_error  : module LCD.C                               *
357   *               xmt_buffer : module BMF.C                               *
358   *               _lcd_state : module LCD.C                               *
359   *                                                                       *
360   * Edit History: 07/08/90 - Written by Robin T. Laird.                   *
361   *                                                                       *
362   \*************************************************************************/
363
364   void lcd_transmit_packet(p, retry)
365   lcd_packet p;
```



```
366   word retry;
367   {
368     lcd_error = AOK;                        /* Assume function successful...    */
369
370     /* Insert packet into transmit buffer and send it.                          */
371     /* If the transmit buffer is empty then the transmitter is disbaled:        */
372     /*   Reset the transmitter source address and byte count.                   */
373     /*   Re-start the transmitter (it was turned off when buf empty).           */
374     /* Else, just insert packet into buffer for transmission (sometime later).*/
375     /* If the insert failed, abort transmission and return an error.            */
376
377     if (buf_empty(&xmt_buffer))
378     {
379       buf_insert(&xmt_buffer, p);
380       if (lcd_error != AOK)
381       {
382         lcd_error = LCD_ERR_TRANSMIT_PACKET;
383         return;
384       }
385       else lcc_start_xmt();
386     }
387     else
388     {
389       buf_insert(&xmt_buffer, p);
390       if (lcd_error != AOK)
391       {
392         lcd_error = LCD_ERR_TRANSMIT_PACKET;
393         return;
394       }
395     }
396
397     /* At a later date, it will be possible to add a packet to the transmit   */
398     /* buffer and then return to the calling function immediately, the packet */
399     /* would be transmitted when it moved to the front of the buffer.         */
400     /* A retry of zero would indicate that the packet is to be transmitted  */
401     /* in the above manner (i.e., don't wait for packet to be transmitted).   */
402
403     /* For now, always wait for completion.                                   */
404
405     if (retry == LCD_DONT_WAIT)
406     {
407       lcd_error = LCD_ERR_TRANSMIT_PACKET;
408     }
409     else if (retry == LCD_WAIT_FOREVER)
410     {
411       while (!buf_empty(&xmt_buffer));
412     }
413     else if (retry <= LCD_MAX_ATTEMPTS)
414     {
415       while (!buf_empty(&xmt_buffer))
416         if (retry-- == 0)
417         {
418           lcd_error = LCD_ERR_TRANSMIT_PACKET;
419           return;
420         }
421     }
422     else
423     {
424       lcd_error = LCD_ERR_NUM_ATTEMPTS;
425     }
426
427     /* If an error occured while inserting packet into buffer, log type.      */
428     /* If no error ocurred, log successful transmission.                      */
429
430     if (lcd_error != AOK)
431       lcd_state.x_err_cnt++;
432     else
433       _lcd_state.x_valid_cnt++;
434   }
435
436
437   /*************************************************************************\
438   *                                lcd_status                               *
```

```
439  *****************************************************************************
440  *                                                                           *
441  * Function:     Returns the operational status of the LCD subsystem.        *
442  *               This is accomplished by returning the contents of the       *
443  *               module state structure that records various operational     *
444  *               parameters such as the number of valid packets received, etc.*
445  *                                                                           *
446  * Input:        lcd_status(                                                 *
447  *                 lcd_state *s  pointer to module state structure.          *
448  *               );                                                          *
449  *                                                                           *
450  * Output:       Nothing.                                                    *
451  *                                                                           *
452  * Globals:      lcd_error : module LCD.C                                    *
453  *                                                                           *
454  * Edit History: 07/09/90 - Written by Richard P. Smurlo.                    *
455  *                                                                           *
456  \*****************************************************************************/
457
458  void lcd_status(s)
459  lcd_state *s;
460  {
461    lcd_error = AOK;                        /* Assume function successful...    */
462
463    *s = _lcd_state;
464  }
```

```
1   /*****************************************************************\
2   *                              LCI.H                              *
3   *****************************************************************
4   *                                                                *
5   * CPCI:          IED90-MRA-COM-LCS-LCI-H-R0C0                    *
6   *                                                                *
7   * Description:   LCS communications interface variables and      *
8   *                functions. Contains constant function parameter *
9   *                declarations as well as function return values  *
10  *                (for success and failure of all operations).    *
11  *                Contains the function prototypes for the LCI.C  *
12  *                module.                                         *
13  *                                                                *
14  *                Module LCI exports the following types/variables/functions:
15  *                                                                *
16  *                typedef lci_message;                            *
17  *                                                                *
18  *                int lci_error;                                  *
19  *                                                                *
20  *                lci_init();                                     *
21  *                lci_receive_message();                          *
22  *                lci_send_message();                             *
23  *                                                                *
24  * Notes:         1) See SDS pp. 5-6 through 5-x for more information.
25  *                                                                *
26  * Edit History: 08/14/90 - Written by Robin T. Laird.            *
27  *                                                                *
28  \*****************************************************************/
29
30  /* Public Data Structures:                                       */
31
32  #ifndef LCI_MODULE_CODE
33  #define LCI_MODULE_CODE          4000
34
35  #define LCI_ERR_NOT_INIT         1+LCI_MODULE_CODE
36  #define LCI_ERR_RECEIVE_MESSAGE  2+LCI_MODULE_CODE
37  #define LCI_ERR_SEND_MESSAGE     3+LCI_MODULE_CODE
38
39  /* Maximum and minimum retry values for send/receive of messages.*/
40  /* Values must correspond with related definitions in module LCD.H.*/
41
42  #define LCI_WAIT_FOREVER         65535
43  #define LCI_DONT_WAIT            0
44  #define LCI_MAX_ATTEMPTS         60000
45
46  /* Maximum message length SHOULD be two less than maximum frame length.*/
47
48  #define LCI_MAX_MESSAGE_LENGTH   SYS_MAX_PACKET_SIZE
49
50  /* MRA inter-module message type defined as a sequence of bytes. */
51
52  typedef byte lci_message[LCI_MAX_MESSAGE_LENGTH];
53
54  /* External module global error variable.                        */
55
56  extern int lci_error;
57
58  /* Public Functions:                                             */
59
60  void lci_init();
61  void lci_receive_message(lci_message m, word retry);
62  void lci_send_message(lci_message m, word retry);
63
64  #endif
```

```
1   /*****************************************************************************\
2   *                                  LCI.C                                      *
3   *****************************************************************************
4   *                                                                            *
5   * CPCI:          IED90-MRA-COM-LCS-LCI-C-R0C0                                *
6   *                                                                            *
7   * Description:   LCS communications interface functions.                     *
8   *                Implements the MRA standard Local Communications            *
9   *                Interface (LCI) module. This module contains the standard   *
10  *                communications interface functions that provide higher-level *
11  *                software access to the Local Communications Channel of the  *
12  *                host processing system (ICN or MPU).                        *
13  *                                                                            *
14  *                Message format at this level (ISO OSI network layer) is:    *
15  *                                                                            *
16  *                 | byte 0 | byte 1 | byte 2 | byte 3 | byte 4 | byte n|      *
17  *                 |-------------------------------------------------|        *
18  *                 |  DEST  | LENGTH | SOURCE |  xxxx  |  xxxx  |  .... |     *
19  *                                                                            *
20  *                Module LCI exports the following variables/functions:       *
21  *                                                                            *
22  *                int lci_error;                                              *
23  *                                                                            *
24  *                lci_init();                                                 *
25  *                lci_receive_message();                                      *
26  *                lci_send_message();                                         *
27  *                                                                            *
28  * Notes:         1) The LCI functions are implementation independent.        *
29  *                2) Module LCI represents the MRA Communications Level.      *
30  *                                                                            *
31  * Edit History: 08/14/90 - Written by Robin T. Laird.                        *
32  *                                                                            *
33  \*****************************************************************************/
34
35  #include <sysdefs.h>                    /* System constants and types.      */
36  #include "lcd.h"                        /* LCD public literals/functions.   */
37  #include "lci.h"                        /* LCI public literals/functions.   */
38
39  /* Public Variables:                                                        */
40
41  /* Global module error variable, lci_error.                                */
42  /* lci_error contains code of last error occurence.                         */
43  /* Should be set to AOK after each successful function call.                */
44  /* Variable can be examined by other software after each function call.     */
45
46  XDATA int lci_error = LCI_ERR_NOT_INIT; /* Global module error variable.    */
47
48
49  /*****************************************************************************\
50  *                                 lci_init                                   *
51  *****************************************************************************
52  *                                                                            *
53  * Function:      Initializes the Local Communications Interface.             *
54  *                The Local Communications Devie Handler (LCD) subsystem      *
55  *                along with all module variables are initialized. Any errors *
56  *                are examined for severity and an attempt is made to recover *
57  *                from non-fatal conditions. If initialization is unsuccessful *
58  *                then the error LCI_ERR_NOT_INIT is returned in lci_error.   *
59  *                                                                            *
60  * Input:         lci_init();                                                 *
61  *                                                                            *
62  * Output:        Nothing.                                                    *
63  *                                                                            *
64  * Globals:       lci_error : module LCI.C                                    *
65  *                lcd_error : module LCD.C                                    *
66  *                                                                            *
67  * Edit History: 07/28/90 - Written by Robin T. Laird.                        *
68  *                                                                            *
69  \*****************************************************************************/
70
71  void lci_init()
72  {
73    lci_error = AOK;                      /* Assume function successful...    */
```



```
74
75    /* Initialize the LCD subsystem (sets up LSC/LPC hardware and software).  */
76
77    lcd_init();
78    if (lcd_error != AOK)
79    {
80      switch(lcd_error)
81      {
82        case LCD_FAIL_RECEIVER:
83        case LCD_FAIL_TRANSMITTER:
84          lci_error = LCI_ERR_NOT_INIT;
85          break;
86        default:
87          lci_error = lcd_error;
88      }
89    }
90  }
91
92
93  /*****************************************************************************\
94  *                            lci_receive_message                             *
95  *****************************************************************************
96  *                                                                            *
97  * Function:      Receives the latest (oldest) message from the LSC/LPC.      *
98  *                If a message is not immediately available, the function     *
99  *                waits for a specific period of time for an incoming message, *
100 *                and then returns regardless. If a message is never received, *
101 *                then the global variable lci_error is set to the literal    *
102 *                LCI_ERR_RECEIVE_MESSAGE. If a message is available then it  *
103 *                is returned immediately.                                    *
104 *                                                                            *
105 *                The number of receive errors is tracked so that if it       *
106 *                exceeds a maximum value over time, the LCS device handler   *
107 *                is reset to try and remedy the problem.                     *
108 *                                                                            *
109 * Input:         lci_receive_message(                                        *
110 *                  lci_message m; received message.                          *
111 *                  word retry;    number of times to try receiving message.  *
112 *                );                                                          *
113 *                                                                            *
114 * Output:        Nothing.                                                    *
115 *                                                                            *
116 * Globals:       lci_error : module LCI.C                                    *
117 *                lcd_error : module LCD.C                                    *
118 *                                                                            *
119 * Edit History: 08/11/90 - Written by Robin T. Laird.                        *
120 *                                                                            *
121 \*****************************************************************************/
122
123 #define MAX_RECV_ERRORS          1000    /* On > 1000 errors, reset LCD.     */
124
125 void lci_receive_message(m, retry)
126 lci_message m;
127 word retry;
128 {
129   lcd_state status;                      /* Holds LCD status.                */
130
131   lci_error = AOK;                       /* Assume function successful...    */
132
133   /* Request packet. Iterate retry number of times.                          */
134
135   lcd_receive_packet(m, retry);
136
137   /* If a packet is not available, check integrity of receiver.              */
138   /* If number of receive errors is excessive then reset the LCD subsystem. */
139   /* If a packet is available then return it.                                */
140
141   if (lcd_error != AOK)
142   {
143     lcd_status(&status);
144     if (status.r_err_cnt > MAX_RECV_ERRORS) lcd_reset();
145     lci_error = LCI_ERR_RECEIVE_MESSAGE;
146   }
```

```
147   }
148
149
150   /*************************************************************************\
151   *                              lci_send_message                           *
152   *************************************************************************
153   *                                                                         *
154   * Function:     Sends the parameter message to the LSC/LPC.               *
155   *               If the message cannot be sent immediately, the function   *
156   *               waits a specific period of time for the transmission and  *
157   *               then returns regardless. If the message is never sent, then *
158   *               the global variable lci_error is set to LCI_ERR_SEND_MESSAGE.*
159   *                                                                         *
160   *               The number of send errors is tracked so that if it        *
161   *               exceeds a maximum value over time, the LCS device handler *
162   *               is reset to try and remedy the problem.                   *
163   *                                                                         *
164   * Input:        lci_send_message(                                         *
165   *                 lci_message m; message to be sent.                      *
166   *                 word retry;    number of times to try sending message.  *
167   *               );                                                        *
168   *                                                                         *
169   * Output:       Nothing.                                                  *
170   *                                                                         *
171   * Globals:      lci_error : module LCI.C                                  *
172   *               lcd_error : module LCD.C                                  *
173   *                                                                         *
174   * Edit History: 08/11/90 - Written by Robin T. Laird.                     *
175   *                                                                         *
176   \*************************************************************************/
177
178   #define MAX_SEND_ERRORS         1000     /* > 1000 errors reset LCD.       */
179
180   void lci_send_message(m, retry)
181   lci_message m;
182   word retry;
183   {
184     lcd_state status;                      /* Holds LCD status.              */
185
186     lci_error = ACK;                       /* Assume function successful...  */
187
188     /* Send packet. Iterate retry number of times.                           */
189
190     lcd_transmit_packet(m, retry);
191
192     /* If packet could not be transmitted, check integrity of transmitter.   */
193     /* If number of transmit errors is excessive then reset the LCD subsystem.*/
194
195     if (lcd_error != AOK)
196     {
197       lcd_status(&status);
198       if (status.x_err_cnt > MAX_SEND_ERRORS) lcd_reset();
199       lci_error = LCI_ERR_SEND_MESSAGE;
200     }
201   }
```

```
1   #*****************************************************************************#
2   #                                  MAKEFILE                                   #
3   #*****************************************************************************#
4   #                                                                             #
5   # CPCI:          IED90-MRA-COM-MMS-MAKEFILE-TXT-R0C0                          #
6   #                                                                             #
7   # Description:   Makefile for the Modular Robotic Architecture (MRA).        #
8   #                Makes the common method manager subsystem.                   #
9   #                                                                             #
10  #                Targets are available for the following systems/subsystems:  #
11  #                                                                             #
12  #                mms    - COM Method Manager Subsystem                        #
13  #                lib    - Add modules to MRA library                          #
14  #                print  - Print COM method manager files                      #
15  #                                                                             #
16  # Notes:         1) The dependency and production rules are included here.    #
17  #                2) See also \mra\makefile.                                   #
18  #                                                                             #
19  # Edit History: 03/22/91 - Written by Robin T. Laird.                         #
20  #                                                                             #
21  #*****************************************************************************#
22
23
24
25  ################################### RULES ###################################
26
27  .SUFFIXES : .hex .exe .obj .c .a51
28
29  # Control settings for Franklin 8031 development
30
31  CC          =c51
32  AS          =a51
33  LINK        =151
34  OTOH        =ohs51
35  CFLAGS      =cd la db sb
36  ASFLAGS     =
37  LFLAGS      =
38  OFLAGS      =
39  STARTUP     =\c51\crom.obj
40  CODESEG     =00000h
41  XDATASEG    =00000h
42
43  # Control settings for Microsoft MS-DOS development
44
45  MSC         =cl
46  MSAS        =masm
47  MSLINK      =link
48  MSCFLAGS    =/AL /c /Oi /Zi /Od
49  MSASFLAGS   =
50  MSLNKFLGS   =/co
51  LOADLIBES   =
52
53  .c.obj :
54          $(CC) $< $(CFLAGS)
55
56  .a51.obj :
57          $(AS) $<  $(ASFLAGS)
58
59  .obj.exe :
60          $(LINK) $(STARTUP),$< TO $@ code ($(CODESEG)) xdata ($(XDATASEG)) ixref
61
62  .exe.hex :
63          $(OTOH) $< $(OFLAGS)
64
65
66
67  ################################## DEFINITIONS ##################################
68
69  # Project, system, and application level definitions
70
71  PROJ            = mra
72  APPSYS          = app
73  COMSYS          = ccm
```



```
74  ICNSYS          = icn
75  MPUSYS          = mpu
76
77  COMLIB          = \$(PROJ)\lib
78  COMSRC          = \$(PROJ)\$(COMSYS)\src
79  COMBIN31        = \$(PROJ)\$(COMSYS)\bin\8031
80  COMBIN152       = \$(PROJ)\$(COMSYS)\bin\80152
81  COMBINMS        = \$(PROJ)\$(COMSYS)\bin\msdos
82  COMBINSBC8      = \$(PROJ)\$(COMSYS)\bin\sbc8
83
84  # Common subsystem level source directories
85
86  HDRSRC          = $(COMSRC)\hdr
87  LCSSRC          = $(COMSRC)\lcs
88  MMSSRC          = $(COMSRC)\mms
89
90  # Common subsystem global include & compilation units
91
92  SYSDEFS         = $(HDRSRC)\sysdefs.h
93
94  MMS             = $(COMBIN152)\mm.obj          $(COMBIN152)\pb.obj      \
95                    $(COMBIN152)\lmmdct.obj      $(COMBIN152)\smmlib.obj \
96                    $(COMBIN31)\mm.obj           $(COMBIN31)\pb.obj       \
97                    $(COMBIN31)\lmmdct.obj       $(COMBIN31)\smmlib.obj   \
98                    $(COMBINMS)\mm.obj           $(COMBINMS)\pb.obj       \
99                    $(COMBINMS)\lmmdct.obj       $(COMBINMS)\smmlib.obj
100
101
102
103 ################################## TARGETS ##################################
104
105 mms             : $(MMS)
106
107 lib             : $(MMS)
108                   -lib51 delete $(COMLIB)\mra_1521.lib (mm, pb, lmmdct, smmlib)
109                   -lib51 add $(COMBIN152)\mm.obj to $(COMLIB)\mra_1521.lib
110                   -lib51 add $(COMBIN152)\pb.obj to $(COMLIB)\mra_1521.lib
111                   -lib51 add $(COMBIN152)\lmmdct.obj to $(COMLIB)\mra_1521.lib
112                   -lib51 add $(COMBIN152)\smmlib.obj to $(COMLIB)\mra_1521.lib
113                   -lib51 delete $(COMLIB)\mra_311.lib (mm, pb, lmmdct, smmlib)
114                   -lib51 add $(COMBIN31)\mm.obj to $(COMLIB)\mra_311.lib
115                   -lib51 add $(COMBIN31)\pb.obj to $(COMLIB)\mra_311.lib
116                   -lib51 add $(COMBIN31)\lmmdct.obj to $(COMLIB)\mra_311.lib
117                   -lib51 add $(COMBIN31)\smmlib.obj to $(COMLIB)\mra_311.lib
118                   -lib $(COMLIB)\mra_mss.lib -mm.obj+$(COMBINMS)\mm.obj;
119                   -lib $(COMLIB)\mra_msl.lib -mm.obj+$(COMBINMS)\mm.obj;
120                   -lib $(COMLIB)\mra_mss.lib -pb.obj+$(COMBINMS)\pb.obj;
121                   -lib $(COMLIB)\mra_msl.lib -pb.obj+$(COMBINMS)\pb.obj;
122                   -lib $(COMLIB)\mra_mss.lib -lmmdct.obj+$(COMBINMS)\lmmdct.obj
123                   -lib $(COMLIB)\mra_msl.lib -lmmdct.obj+$(COMBINMS)\lmmdct.obj
124                   -lib $(COMLIB)\mra_mss.lib -smmlib.obj+$(COMBINMS)\smmlib.obj
125                   -lib $(COMLIB)\mra_msl.lib -smmlib.obj+$(COMBINMS)\smmlib.obj
126                   touch lib
127
128 print           : $(MMS)
129                   %-a2ps -nf lmm.h | post
130                   %-a2ps -nf lmm.c | post
131                   %-a2ps -nf lmmdct.c | post
132                   %-a2ps -nf mm.h | post
133                   %-a2ps -nf mm.c | post
134                   %-a2ps -nf pb.h | post
135                   %-a2ps -nf pb.c | post
136                   %-a2ps -nf smm.h | post
137                   %-a2ps -nf smm.c | post
138                   %-a2ps -nf smmlib.c | post
139                   touch print
140
141
142
143 ############################## COM MMS DEPENDENCIES ##############################
144
145 # COM Local Method Manager dictionary module dependencies
146
```

```
147  LMMDCT = $(SYSDEFS) $(MMSSRC)\mm.h $(MMSSRC)\lmm.h $(MMSSRC)\lmmdct.c
148
149  $(COMBIN152)\lmmdct.obj          : $(LMMDCT)
150    $(CC) $(MMSSRC)\$*.c $(CFLAGS) df(180152) pr($(MMSSRC)\$*.152) oj($(COMBIN152
151
152  $(COMBIN31)\lmmdct.obj           : $(LMMDCT)
153    $(CC) $(MMSSRC)\$*.c $(CFLAGS) df(18031) pr($(MMSSRC)\$*.31) oj($(COMBIN31)\$
154
155  $(COMBINMS)\lmmdct.obj           : $(LMMDCT)
156    $(MSC) $(MSCFLAGS) /DIBMAT /Fs$(MMSSRC)\$*.at /Fo$(COMBINMS)\$* $(MMSSRC)\$*.
157
158
159  # COM System Method Manager library module dependencies
160
161  SMMLIB = $(SYSDEFS) $(MMSSRC)\mm.h $(MMSSRC)\smm.h $(MMSSRC)\smmlib.c
162
163  $(COMBIN152)\smmlib.obj          : $(SMMLIB)
164    $(CC) $(MMSSRC)\$*.c $(CFLAGS) df(180152) pr($(MMSSRC)\$*.152) oj($(COMBIN152
165
166  $(COMBIN31)\smmlib.obj           : $(SMMLIB)
167    $(CC) $(MMSSRC)\$*.c $(CFLAGS) df(18031) pr($(MMSSRC)\$*.31) oj($(COMBIN31)\$
168
169  $(COMBINMS)\smmlib.obj           : $(SMMLIB)
170    $(MSC) $(MSCFLAGS) /DIBMAT /Fs$(MMSSRC)\$*.at /Fo$(COMBINMS)\$* $(MMSSRC)\$*.
171
172
173  # COM Phone Book module dependencies
174
175  PB = $(SYSDEFS) $(MMSSRC)\mm.h $(MMSSRC)\smm.h $(MMSSRC)\pb.h $(MMSSRC)\pb.c
176
177  $(COMBIN152)\pb.obj              : $(PB)
178    $(CC) $(MMSSRC)\$*.c $(CFLAGS) df(180152) pr($(MMSSRC)\$*.152) oj($(COMBIN152
179
180  $(COMBIN31)\pb.obj               : $(PB)
181    $(CC) $(MMSSRC)\$*.c $(CFLAGS) df(18031) pr($(MMSSRC)\$*.31) oj($(COMBIN31)\$
182
183  $(COMBINMS)\pb.obj               : $(PB)
184    $(MSC) $(MSCFLAGS) /DIBMAT /Fs$(MMSSRC)\$*.at /Fo$(COMBINMS)\$* $(MMSSRC)\$*.
185
186
187  # COM Method Manager module dependencies
188
189  MM = $(SYSDEFS)          $(LCSSRC)\lc1.h          $(MMSSRC)\lmm.h          \
190       $(MMSSRC)\smm.h     $(MMSSRC)\mm.h           $(MMSSRC)\lmm.c          \
191       $(MMSSRC)\smm.c     $(MMSSRC)\mm.c
192
193  $(COMBIN152)\mm.obj              : $(MM)
194    $(CC) $(MMSSRC)\$*.c $(CFLAGS) df(180152) pr($(MMSSRC)\$*.152) oj($(COMBIN152
195
196  $(COMBIN31)\mm.obj               : $(MM)
197    $(CC) $(MMSSRC)\$*.c $(CFLAGS) df(18031) pr($(MMSSRC)\$*.31) oj($(COMBIN31)\$
198
199  $(COMBINMS)\mm.obj               : $(MM)
200    $(MSC) $(MSCFLAGS) /DIBMAT /Fs$(MMSSRC)\$*.at /Fo$(COMBINMS)\$* $(MMSSRC)\$*.
```

```
1   /*****************************************************************************\
2   *                                  LMM.H                                      *
3   *****************************************************************************
4   *                                                                            *
5   * CPCI:          1ED90-MRA-COM-MMS-LMM-H-R0C0                                *
6   *                                                                            *
7   * Description:   Local Method Manager (LMM) variables and functions.         *
8   *                Contains constant function parameter declarations (#defines) *
9   *                as well as function return values (for success and failure  *
10  *                of all operations). Contains the function prototypes for the *
11  *                LMM.C module.                                               *
12  *                                                                            *
13  *                Defines the dictionary functions for the inherited classes. *
14  *                Each .DCT file defines the dictionary for that MODBOT unit. *
15  *                The functions in the dictionary represent a unit's methods  *
16  *                that are available to other units in the MODBOT system.     *
17  *                Only one module-specific dictionary is valid at a time.     *
18  *                                                                            *
19  *                Module LMM exports the following variables/functions:       *
20  *                                                                            *
21  *                int lmm_error;                                              *
22  *                                                                            *
23  *                lmm_init();                                                 *
24  *                lmm_process();                                              *
25  *                lmm_fire();                                                 *
26  *                lmm_generate_message();                                     *
27  *                lmm_translate_message();                                    *
28  *                                                                            *
29  *                int lmm_num_funcs;                                          *
30  *                int (*lmm_dictionary[]) ();                                 *
31  *                                                                            *
32  *                v_obj_class[];                                              *
33  *                v_obj_superclass[];                                         *
34  *                                                                            *
35  *                v_unit_name[];                                              *
36  *                v_unit_reset;                                               *
37  *                                                                            *
38  * Notes:         1) See SDS pp. 5-6 through 5-x for more information.        *
39  *                                                                            *
40  * Edit History: 10/01/90 - Written by Robin T. Laird.                        *
41  *                                                                            *
42  \*****************************************************************************/
43
44  #ifndef LMM_MODULE_CODE
45  #define LMM_MODULE_CODE                 8000
46
47  /* Public Data Structures:                                                   */
48
49  #define LMM_ERR_NOT_INIT                1+LMM_MODULE_CODE
50  #define LMM_ERR_MSG_TRANSLATION         2+LMM_MODULE_CODE
51  #define LMM_ERR_MSG_GENERATION          3+LMM_MODULE_CODE
52
53  /* External module global error variable.                                   */
54
55  extern int lmm_error;
56
57  /* Public Functions:                                                         */
58
59  void lmm_init();
60  void lmm_process(mm_message *m_in, mm_message *m_out, byte *status);
61  void lmm_fire(void);
62  void lmm_generate_message(mm_message *m_in, mm_message *m_out, byte *status);
63  void lmm_translate_message(mm_message *m_in, mm_message *m_out, byte *status);
64
65  /* Object class literals.                                                    */
66
67  #define OBJ_OBJECT_CLASS        "OBJECT"            /* class names          */
68  #define OBJ_MODULE_CLASS        "MODULE"
69
70  /* Object class methods and instance variables.                              */
71
72  int d_obj_class(), d_obj_superclass();
73  extern char v_obj_class[];
```



```
74  extern char v_obj_superclass[];
75
76  /* Unit class literals.                                                      */
77
78  #define UNIT_NOT_INIT           1                   /* unit status       */
79  #define UNIT_IDLE               2
80  #define UNIT_OFF_LINE           3
81
82  /* Unit class methods and instance variables.                                */
83
84  int d_unit_name(), d_unit_reset();
85  extern char v_unit_name[];
86  extern int  v_unit_reset;
87
88  /* Indicates the total number of functions in the dictionary.                */
89  /* Variable is declared and assigned a value in each individual unit.        */
90
91  extern int lmm_num_funcs;
92
93  /* Unit dictionary is an array of function (method) pointers.                */
94  /* Variable is declared and initialized in each individual unit.             */
95
96  extern int (*lmm_dictionary[]) ();
97
98  #endif
```

```
1   /*****************************************************************************\
2   *                                  LMM.C                                      *
3   ******************************************************************************
4   *                                                                             *
5   * CPCI:          IED90-MRA-COM-MMS-LMM-C-R0C0                                 *
6   *                                                                             *
7   * Description:   MMS local method manager functions.                          *
8   *                Implements the MRA standard Local Method Manager (LMM)       *
9   *                module. This module contains the functions required to       *
10  *                process incoming messages that represent local method        *
11  *                activation requests from external processes. It manages      *
12  *                the MPU Dictionary data structure that contains available    *
13  *                local methods as executable functions.                       *
14  *                                                                             *
15  *                Module LMM exports the following variables/functions:        *
16  *                                                                             *
17  *                int lmm_error;                                               *
18  *                                                                             *
19  *                lmm_init();                                                  *
20  *                lmm_process();                                               *
21  *                lmm_fire();                                                  *
22  *                lmm_generate_message();                                      *
23  *                lmm_translate_message();                                     *
24  *                                                                             *
25  * Notes:         1) The LMM functions are implementation independent.         *
26  *                2) This module is NOT a stand-alone compilation unit.        *
27  *                   It is included by the module MM.C and is compiled there.  *
28  *                   It is assumed that the file LMM.H is included before it.  *
29  *                                                                             *
30  * Edit History: 12/20/90 - Written by Robin T. Laird.                         *
31  *                                                                             *
32  \*****************************************************************************/
33
34  /* Public Variables:                                                        */
35
36  /* Global module error variable, lmm_error.                                 */
37  /* lmm_error contains code of last error occurence.                         */
38  /* Should be set to AOK after each successful function call.                */
39  /* Variable can be examined by other software after each function call.     */
40
41  XDATA int lmm_error = LMM_ERR_NOT_INIT; /* Global module error variable.    */
42
43  /* Local method manager trigger condition table.                            */
44  /* Holds method activation conditions and corresponding method to activate. */
45  /* Var lmm_num_methods holds count of methods in trigger condition table.   */
46
47  #define LMM_MAX_METHODS          1       /* Number of methods we can hold.  */
48
49  typedef struct { mm_message    method;
50                   word          period;
51                   unsigned long fireat;
52                 } lmm_cond;
53
54  static XDATA int lmm_num_methods = 0;
55  static XDATA lmm_cond lmm_trigger[LMM_MAX_METHODS];
56
57
58  /*****************************************************************************\
59  *                                 lmm_init                                    *
60  ******************************************************************************
61  *                                                                             *
62  * Function:      Initializes the Local Method Manager software subsystems.    *
63  *                This includes initializing the module-specific subsystems    *
64  *                via a call to the system function d_unit_reset().            *
65  *                                                                             *
66  * Input:         lmm_in t();                                                  *
67  *                                                                             *
68  * Output:        Nothing.                                                     *
69  *                                                                             *
70  * Globals:       lmm_error       : LMM.C                                      *
71  *                lmm_num_methods : LMM.C                                      *
72  *                                                                             *
73  * Edit History: 12/20/90 - Written by Robin T. Laird.                         *
```



```
74  *                                                                             *
75  \*****************************************************************************/
76
77  void lmm_init()
78  {
79    lmm_error = AOK;                        /* Assume function successful.    */
80
81    /* Reset local method trigger table index to 0 (indicate table empty).    */
82
83    lmm_num_methods = 0;
84
85    /* Call module reset dictionary function (module specific initialization).*/
86
87    d_unit_reset();
88  }
89
90
91  /*****************************************************************************\
92  *                                lmm_process                                  *
93  ******************************************************************************
94  *                                                                             *
95  * Function:      Processes the parameter message and updates the local method *
96  *                trigger table. Coordinates message translation and message   *
97  *                generation. The input message is processed (translated) and  *
98  *                any possible output or response message is generated for     *
99  *                transmission to the originating unit.                        *
100 *                                                                             *
101 * Input:         lmm_process(                                                 *
102 *                  mm_message *m_in;  pointer to decoded message to process.  *
103 *                  mm_message *m_out; pointer to possible message to output.  *
104 *                  byte *status;      pointer to method execution status.     *
105 *                );                                                           *
106 *                                                                             *
107 * Output:        Nothing.                                                     *
108 *                                                                             *
109 * Globals:       lmm_error       : LMM.C                                      *
110 *                lmm_trigger     : LMM.C                                      *
111 *                lmm_num_methods : LMM.C                                      *
112 *                                                                             *
113 * Edit History: 12/20/90 - Written by Robin T. Laird.                         *
114 *                                                                             *
115 \*****************************************************************************/
116
117 void lmm_process(m_in, m_out, status)
118 mm_message *m_in, *m_out;
119 byte *status;
120 {
121   word period;
122
123   lmm_error = AOK;                          /* Assume function successful... */
124
125   /* Check transaction category for periodic method activation.             */
126
127   switch(m_in->trans_category)
128   {
129     case MM_PERIODIC_STATUS_REQUEST:
130
131       /* Extract period from parameter field.                               */
132       /* Period is one word long (ranges from 0 - 65535 ms).                */
133
134       period = (word)m_in->parameter[MM_PPOS] << 8 | m_in->parameter[MM_PPOS+1];
135
136       /* Shift remaining parameter bytes to reflect removal of period.      */
137       /* This is done so that the local method doesn't see period.          */
138
139       m_in->parameter_length -= MM_PLEN;
140       memmove(m_in->parameter,
141               &m_in->parameter[MM_PPOS+MM_PLEN],
142               m_in->parameter_length);
143
144       /* If the period is zero, remove method from trigger table.           */
145       /* Currently, removing the method requires decrementing the index.    */
146       /* Later, we'll have to find the method first and then delete.        */
```

```
147         /* The method would be looked up according to function and sequence #.*/
148
149         if (period == 0)
150         {
151           lmm_num_methods = (lmm_num_methods ? lmm_num_methods-1 : 0);
152           return;
153         }
154         else if (lmm_num_methods < LMM_MAX_METHODS)
155         {
156           /* Add method info to trigger table.                              */
157
158           lmm_trigger[lmm_num_methods].method = *m_in;
159           lmm_trigger[lmm_num_methods].period = period;
160           lmm_trigger[lmm_num_methods].fireat = period+rtc_time();
161           lmm_num_methods++;
162         }
163         break;
164
165       default:
166         break;
167     }
168
169     /* Translate the message (activate function).                         */
170     /* Input message, m_in,  contains translation information.            */
171     /* Output message, m_out, contains any results (in the parameter field).  */
172
173     lmm_translate_message(m_in, m_out, status);
174
175     /* Generate any required response using data from input message.      */
176     /* Input message, m_in, contains data concerning required response.   */
177     /* Output message, m_out, contains function results and message ACK, etc. */
178
179     lmm_generate_message(m_in, m_out, status);
180   }
181
182
183   /*****************************************************************************\
184   *                                lmm_fire                                   *
185   *****************************************************************************
186   *                                                                           *
187   * Function:     Checks local method trigger table and fires appropriate     *
188   *               functions according to conditions set up in the table.      *
189   *               The conditions are set by external commands received from   *
190   *               other modules. The conditions are checked as often as       *
191   *               possible for possible execution of a function.              *
192   *                                                                           *
193   *               Currently, only temporal conditions are implemented.        *
194   *               This allows for periodic execution of functions.            *
195   *                                                                           *
196   * Input:        lmm_fire();                                                 *
197   *                                                                           *
198   * Output:       Nothing.                                                    *
199   *                                                                           *
200   * Globals:      lmm_error       : LMM.C                                     *
201   *               lmm_trigger     : LMM.C                                     *
202   *               lmm_num_methods : LMM.C                                     *
203   *                                                                           *
204   * Edit History: 03/27/91 - Written by Robin T. Laird.                       *
205   *                                                                           *
206   \*****************************************************************************/
207
208   #define LMM_SEND_ATTEMPTS       LCI_WAIT_FOREVER
209
210   void lmm_fire()
211   {
212     int         i;
213     byte        status;
214     lci_message l_out;
215     mm_message  m_out;
216
217     lmm_error = AOK;                    /* Assume function successful...  */
218
219     /* Check number of methods in trigger table. If non-zero, continue.   */
```



```
220
221     if (lmm_num_methods)
222     {
223       /* Check each method for activation.                                */
224       /* Compare current time to trigger time. If greater, then fire method.  */
225
226       for (i = 0; i < lmm_num_methods; i++)
227       {
228         if (rtc_time() > lmm_trigger[i].fireat)
229         {
230           /* Process method as usual.                                     */
231           /* Update next fire time.                                       */
232
233           lmm_trigger[i].fireat = lmm_trigger[i].period + rtc_time();
234           lmm_translate_message(&lmm_trigger[i].method, &m_out, &status);
235           lmm_generate_message(&lmm_trigger[i].method, &m_out, &status);
236
237           /* If response required, encode message, and send via LCI.      */
238
239           if (status == MM_RESPONSE_REQUIRED)
240           {
241             mm_encode_message(&m_out, l_out);
242             lci_send_message(l_out, LMM_SEND_ATTEMPTS);
243           }
244         }
245       }
246     }
247   }
248
249
250   /*****************************************************************************\
251   *                              lmm_generate_message                         *
252   *****************************************************************************
253   *                                                                           *
254   * Function:     Generates a response message, m_out, based on the parameter *
255   *               input message, m_in. The transaction category and the       *
256   *               function status indicate how to respond to the input        *
257   *               message. The transaction disposition field is modified to   *
258   *               either command unknown, executed or failure depending on    *
259   *               the function status value. Since the output message is a    *
260   *               response to the input message the source and destination    *
261   *               address are swapped in the output message. Unchanged fields *
262   *               are copied from the input message to the output message.    *
263   *                                                                           *
264   * Input:        lmm_decode(                                                 *
265   *                 mm_message *m_in;  pointer to (old) translated message.   *
266   *                 mm_message *m_out; pointer to response/results message.   *
267   *                 byte *status;      pointer to translated function status. *
268   *               );                                                          *
269   *                                                                           *
270   * Output:       Nothing.                                                    *
271   *                                                                           *
272   * Globals:      lmm_error : LMM.C                                           *
273   *                                                                           *
274   * Edit History: 12/20/90 - Written by Robin T. Laird.                       *
275   *                                                                           *
276   \*****************************************************************************/
277
278   void lmm_generate_message(m_in, m_out, status)
279   mm_message *m_in, *m_out;
280   byte *status;
281   {
282     /* First check and see if the input message requires a response.      */
283     /* Certain transaction categories require responses, others don't.   */
284     /* If status from translated function indicates no response, just return. */
285
286     if (*status==MM_SUPPRESS_OUTPUT || m_in->trans_category==MM_CONTROL_NO_ACK)
287     {
288       *status = MM_NO_RESPONSE;
289       lmm_error = AOK;
290       return;
291     }
292
```

```
293     /* Otherwise we have to generate a response message...                  */
294
295     /* Swap source and destination addresses of output and input messages.  */
296
297     m_out->dest_modbot_id = m_in->src_modbot_id;
298     m_out->dest_unit_id   = m_in->src_unit_id;
299
300     m_out->src_modbot_id  = m_in->dest_modbot_id;
301     m_out->src_unit_id    = m_in->dest_unit_id;
302
303     /* Sequence number of output message remains unchanged.                 */
304
305     m_out->sequence_number = m_in->sequence_number;
306
307     /* Transaction disposition of output message is set to function status. */
308     /* Status is one of:                                                    */
309     /* MM_COMMAND_RECEIVED : Message received OK, no results generated.     */
310     /* MM_COMMAND_EXECUTED : Method executed OK, results in output message. */
311     /* MM_COMMAND_UNKNOWN  : Invalid function ID, no method executed.       */
312     /* MM_COMMAND_EXECUTION_FAILURE : Invalid parameter (bad parameter, etc.).*/
313
314     m_out->trans_disposition = *status;
315
316     /* Transaction category of output message remains unchanged.            */
317
318     m_out->trans_category = m_in->trans_category;
319
320     /* Function ID of output message remains unchanged.                     */
321
322     m_out->function_id = m_in->function_id;
323
324     /* Parameter length and paramater buffer are already in output message. */
325     /* Set status to indicate that a response message is required.          */
326
327     *status = MM_RESPONSE_REQUIRED;
328     lmm_error = AOK;
329   }
330
331
332   /*************************************************************************\
333   *                          lmm_translate_message                        *
334   *************************************************************************
335   *                                                                       *
336   * Function:     Translates the parameter input message (m_in) into a local  *
337   *               method (function) activation based on the function ID. The  *
338   *               parameter passing field is modified to hold function results *
339   *               (if any). The status variable indicates the success of the  *
340   *               activated method. A status value of MM_COMMAND_EXECUTED     *
341   *               means all went well. If the function failed, then the exact *
342   *               reason for failure is encoded in the parameter field as a   *
343   *               byte value. The local method manager global error variable, *
344   *               lmm_error, is set to LMM_ERR_MSG_TRANSLATION if the local   *
345   *               method failed.                                              *
346   *                                                                       *
347   * Input:        lmm_translate_message(                                      *
348   *                 mm_message *m_in;  pointer to message to translate.       *
349   *                 mm_message *m_out; pointer to message holding results.    *
350   *                 byte *status;      pointer to translated function status. *
351   *               );                                                          *
352   *                                                                       *
353   * Output:       Nothing.                                                    *
354   *                                                                       *
355   * Globals:      lmm_error     : LMM.C                                       *
356   *               lmm_num_funcs : DICTIONARY                                  *
357   *                                                                       *
358   * Edit History: 01/04/91 - Written by Robin T. Laird.                       *
359   *                                                                       *
360   \*************************************************************************/
361
362   void lmm_translate_message(m_in, m_out, status)
363   mm_message *m_in, *m_out;
364   byte *status;
365   {
```



```
366     /* If function ID invalid, indicate command unknown and return.        */
367
368     if (m_in->function_id > lmm_num_funcs)
369     {
370       *status   = MM_COMMAND_UNKNOWN;
371       lmm_error = LMM_ERR_MSG_TRANSLATION;
372     }
373     else
374     {
375       /* Copy parameters (if any) to standard parameter input buffer.      */
376       /* Execute function indicated by function ID in input message.       */
377       /* Copy results from standard parameter output buffer to message buffer.*/
378       /* Note that we have to adjust for bit output by testing bit index.  */
379
380       memcpy(mm_stdin.buffer, m_in->parameter, (int)m_in->parameter_length);
381       mm_stdin.index     = 0;
382       mm_stdin.bitindex  = 0;
383       mm_stdout.index    = 0;
384       mm_stdout.bitindex = 0;
385
386       *status = (*lmm_dictionary[m_in->function_id])();
387
388       if (mm_stdout.bitindex) mm_stdout.index++;
389       memcpy(m_out->parameter, mm_stdout.buffer, mm_stdout.index);
390       m_out->parameter_length = mm_stdout.index;
391       mm_stdin.index     = 0;
392       mm_stdin.bitindex  = 0;
393       mm_stdout.index    = 0;
394       mm_stdout.bitindex = 0;
395
396       /* If there is an error in executing the function above, then:       */
397       /*   1. Status variable will be set to MM_COMMAND_EXECUTION_FAILURE. */
398       /*   2. Source of error will be encoded in parameter output buffer.  */
399       /* A NULL status indicates that no response message is to be generated. */
400
401       switch(*status)
402       {
403         case MM_INITIATING:
404         case MM_COMMAND_RECEIVED:
405         case MM_COMMAND_EXECUTED:
406         case MM_SUPPRESS_OUTPUT:
407           lmm_error = AOK;
408           break;
409
410         case MM_COMMAND_UNKNOWN:
411         case MM_COMMAND_EXECUTION_FAILURE:
412           lmm_error = LMM_ERR_MSG_TRANSLATION;
413           break;
414
415         default:
416           break;
417       }
418     }
419   }
```

```
1     /*****************************************************************************\
2     *                                  LMMDCT.C                                   *
3     *****************************************************************************
4     *                                                                            *
5     * CPCI:          IED90-MRA-COM-MMS-LMMDCT-C-R0C0                             *
6     *                                                                            *
7     * Description:   Local Method Manager (LMM) dictionary functions.            *
8     *                Implements the inherited ("built-in") local methods.        *
9     *                                                                            *
10    *                Module LMMDCT.C exports the following variables/functions:  *
11    *                                                                            *
12    *                int lmm_num_funcs;                                          *
13    *                                                                            *
14    *                OBJECT (STATUS_REQUEST):                                    *
15    *                d_obj_class();                                              *
16    *                d_obj_superclass();                                         *
17    *                                                                            *
18    *                UNIT (STATUS_REQUEST):                                      *
19    *                d_unit_name();                                              *
20    *                                                                            *
21    *                UNIT (CONTROL, CONTROL_WITH_ACK):                           *
22    *                d_unit_reset();                                             *
23    *                                                                            *
24    * Notes:         None.                                                       *
25    *                                                                            *
26    * Edit History: 01/09/91 - Written by Robin T. Laird.                        *
27    *                                                                            *
28    \*****************************************************************************/
29
30    #include <sysdefs.h>                    /* System constants and types.      */
31    #include "mm.h"                         /* Method manager literals.         */
32    #include "lmm.h"                        /* Local method manager.            */
33
34    /* Object class methods...                                                   */
35
36    int d_obj_class()
37    {
38      mm_sprintfb(&mm_stdout, "%s", v_obj_class);
39      return(MM_COMMAND_EXECUTED);
40    }
41
42    int d_obj_superclass()
43    {
44      mm_sprintfb(&mm_stdout, "%s", v_obj_superclass);
45      return(MM_COMMAND_EXECUTED);
46    }
47
48    /* Unit class methods...                                                     */
49
50    int d_unit_name()
51    {
52      mm_sprintfb(&mm_stdout, "%s", v_unit_name);
53      return(MM_COMMAND_EXECUTED);
54    }
55
56    /* Prototype is given below (all reset functions return an int).           */
57    /* The reset() function is defined in each MODBOT module dictionary (file). */
58
59    int reset();
60    int d_unit_reset()
61    {
62      v_unit_reset = UNIT_IDLE;
63      reset();
64      return(MM_COMMAND_EXECUTED);
65    }
```

```
/*****************************************************************************\
*                                    MM.H                                     *
*****************************************************************************
*                                                                           *
* CPCI:          IED90-MRA-COM-MMS-MM-H-R0C2                                *
*                                                                           *
* Description:   Method Manager (MM) variables and functions.               *
*                Contains constant function parameter declarations (#defines) *
*                as well as function return values (for success and failure  *
*                of all operations). Contains the function prototypes for the *
*                MM.C module.                                               *
*                                                                           *
*                Module MM exports the following types/variables/functions: *
*                                                                           *
*                typedef mm_pbuffer;                                        *
*                                                                           *
*                int mm_error;                                              *
*                mm_pbuffer mm_stdin;                                       *
*                mm_pbuffer mm_stdout;                                      *
*                                                                           *
*                mm_init();                                                 *
*                mm_cycle();                                                *
*                mm_encode_message();                                       *
*                mm_decode_message();                                       *
*                mm_sprintfb();                                             *
*                mm_sscanfb();                                              *
*                mm_service_event();                                        *
*                mm_terminate_event();                                      *
*                mm_check_event();                                          *
*                                                                           *
* Notes:         1) See SDS pp. 5-6 through 5-x for more information.       *
*                                                                           *
* Edit History: 10/01/90 - Written by Robin T. Laird.                       *
*                                                                           *
\*****************************************************************************/

#ifndef MM_MODULE_CODE
#define MM_MODULE_CODE                  7000

#include <lci.h>                        /* Local communications interface.  */

/* Public Data Structures:                                                  */

#define MM_ERR_NOT_INIT                 1+MM_MODULE_CODE
#define MM_ERR_MESSAGE_NOT_AVAILABLE    2+MM_MODULE_CODE
#define MM_ERR_MESSAGE_DISPOSITION      3+MM_MODULE_CODE
#define MM_ERR_LOCAL_METHOD_INIT        4+MM_MODULE_CODE
#define MM_ERR_SYSTEM_METHOD_INIT       5+MM_MODULE_CODE
#define MM_ERR_IN_PHONEBOOK             6+MM_MODULE_CODE
#define MM_ERR_NO_EVENT_PENDING         7+MM_MODULE_CODE

#define MM_CYCLE_FOREVER                32767
#define MM_RESPONSE_REQUIRED            0
#define MM_NO_RESPONSE                  1

/* Events are assigned unique numbers as identifiers for later cross-ref.   */
/* Invalid method activations return value MM_NULL_EVENT.                   */

#define MM_NULL_EVENT                   -1
#define MM_AWAITING_EVENT               0

/* Inter-module message format (IMMF) (ISO OSI presentation layer).         */
/* IMMF transaction disposition values.                                     */

#define MM_INITIATING                   1
#define MM_COMMAND_RECEIVED             2
#define MM_COMMAND_EXECUTED             3
#define MM_COMMAND_UNKNOWN              4
#define MM_COMMAND_EXECUTION_FAILURE    5
#define MM_SUPPRESS_OUTPUT              6

/* IMMF transaction category values.                                        */
```



```
#define MM_CONTROL_WITH_ACK             1
#define MM_CONTROL_NO_ACK               2
#define MM_STATUS_REQUEST               3
#define MM_PERIODIC_STATUS_REQUEST      4
#define MM_QUERY_CONTROL                5
#define MM_SET_ALARM_LIMITS             6
#define MM_QUERY_ALARM_LIMITS           7
#define MM_SET_OPERATING_LIMITS         8
#define MM_QUERY_OPERATING_LIMITS       9
#define MM_INDICATION                   10
#define MM_ALARM_ACTIVATED              11
#define MM_ALARM_RETIRED                12
#define MM_COMMAND_EXECUTION_INDICATION 13
#define MM_COMMAND_FAILED_INDICATION    14

/* Type definition for IMMF, used to exchange info between functions.       */

#define MM_MAX_PARAM_SIZE               SYS_MAX_PACKET_SIZE

typedef struct { byte dest_modbot_id;
                 byte dest_unit_id;
                 byte message_length;
                 byte src_modbot_id;
                 byte src_unit_id;
                 byte sequence_number;
                 byte trans_disposition;
                 byte trans_category;
                 byte function_id;
                 byte parameter_length;
                 byte parameter[MM_MAX_PARAM_SIZE];
               } mm_message;

/* Type definition for parameter buffer, used for formatted I/O of data.    */
/* Index gives the next byte position within the buffer.                    */
/* Bit index gives next bit position within the current byte position.      */
/* Variables mm_stdin and mm_stdout are available for standard data I/O.    */

#define MM_PBUFFER_SIZE                 MM_MAX_PARAM_SIZE

typedef struct { byte buffer[MM_PBUFFER_SIZE];
                 int  index;
                 byte bitindex;
               } mm_pbuffer;

extern mm_pbuffer mm_stdin, mm_stdout;

/* External module global error variable.                                   */

extern int mm_error;

/* Public Functions:                                                        */

void mm_init();
void mm_cycle(int iterations);
void mm_encode_message(mm_message *to_encode, lci_message encoded);
void mm_decode_message(lci_message to_decode, mm_message *decoded);
void mm_sprintfb(mm_pbuffer *p, char *format, ...);
void mm_sscanfb(mm_pbufier *p, char *format, ...);
void mm_service_event(int event, mm_message *m_out);
void mm_terminate_event(int event);
int  mm_check_event(int event);

#endif
```

```
/*****************************************************************************\
*                                    MM.C                                     *
*****************************************************************************
*                                                                           *
* CPCI:          IED90-MRA-COM-MMS-MM-C-R0C2                                *
*                                                                           *
* Description:   MMS (system) method manager functions.                     *
*                Implements the MRA standard Method Manager (MM) module.    *
*                This module contains the functions that control the local  *
*                and remote method manager subsystems.                      *
*                                                                           *
*                Message format at this level (ISO OSI presentation layer) is:*
*                                                                           *
*                  |byte 0|byte 1|byte 2| byte 3 | byte 4 |byte 5|byte 6|  *
*                  |-----------------------------------------------------|  *
*                  | DEST |      | SORC |         | TRANSC |      |      |  *
*                  |DEVICE| MESG |DEVICE|SEQ NUMB|DISPOST/| FN ID|PARAMS| ... *
*                  |  ID  |LENGTH|  ID  |         | TRANSC |      |LENGTH|  *
*                  |      |      |      |         |CATEGORY|      |      |  *
*                                                                           *
*                Module MM exports the following variables/functions:       *
*                                                                           *
*                int mm_error;                                              *
*                mm_pbuffer mm_stdin;                                       *
*                mm_pbuffer mm_stdout;                                      *
*                                                                           *
*                mm_init();                                                 *
*                mm_cycle();                                                *
*                mm_encode_message();                                       *
*                mm_decode_message();                                       *
*                mm_sprintfb();                                             *
*                mm_sscanfb();                                              *
*                mm_service_event();                                        *
*                mm_terminate_event();                                      *
*                mm_check_event();                                          *
*                                                                           *
* Notes:         1) The MM functions are implementation independent.        *
*                                                                           *
* Edit History: 12/20/90 - Written by Robin T. Laird.                       *
*                                                                           *
\*****************************************************************************/

#include <string.h>                     /* Standard string functions.       */
#include <sysdefs.h>                    /* System constants and types.      */
#include <stdarg.h>                     /* Variable argument macros.        */
#include <rtc.h>                        /* Real-time clock functions.       */
#include "mm.h"                         /* Method manager.                  */
#include "pb.h"                         /* Method manager phone book.       */
#include "lmm.h"                        /* Local method manager.            */
#include "smm.h"                        /* System method manager.           */

#if defined(DEBUG)
#include <debug.h>
#endif

/* Public Variables:                                                        */

/* Global module error variable, mm_error.                                 */
/* mm_error contains code of last error occurence.                          */
/* Should be set to AOK after each successful function call.                */
/* Variable can be examined by other software after each function call.     */

XDATA int mm_error = MM_ERR_NOT_INIT;   /* Global module error variable.    */

/* Global standard input and output parameter (passing) buffers.            */

XDATA mm_pbuffer mm_stdin, mm_stdout;

/* Definitions for field sizes/positions. Maintain with care!               */

#define MM_MODBOT_ID_BIT_SHIFT          5
#define MM_UNIT_ID_MASK                 0x1F
#define MM_TRANS_DISPOSIT_BIT_SHIFT     4
```



```
#define MM_TRANS_CATEGORY_MASK          0x0F

/* Position of period (in ms) in parameter field.                           */
/* Length of period in bytes.                                               */

#define MM_PPOS                         0
#define MM_PLEN                         2

/* Position of destination address in message.                              */
/* Number of overhead bytes at this communications level.                   */

#define MM_DEST_ADDR_POS                0
#define MM_COM_OVERHEAD_BYTES           7

/* Local method manager functions are included here.                        */

#include "lmm.c"

/* System method manager functions are included here.                       */

#include "smm.c"


/*****************************************************************************\
*                                   mm_init                                 *
*****************************************************************************
*                                                                           *
* Function:      Initializes the Local Method Manager software subsystems.  *
*                This includes initializing data structures such as the     *
*                system phone book and the message print I/O buffers.       *
*                                                                           *
* Input:         mm_init();                                                 *
*                                                                           *
* Output:        Nothing.                                                   *
*                                                                           *
* Globals:       mm_error  : MM.C                                           *
*                lmm_error : LMM.C                                          *
*                smm_error : SMM.C                                          *
*                mm_stdin  : MM.C                                           *
*                mm_stdout : MM.C                                           *
*                                                                           *
* Edit History: 12/20/90 - Written by Robin T. Laird.                       *
*                                                                           *
\*****************************************************************************/

void mm_init()
{
  int i;

  /* Initialize the local method manager (LMM).                             */

  lmm_init();
  if (lmm_error != AOK)
  {
    mm_error = MM_ERR_LOCAL_METHOD_INIT;
    return;
  }

  /* Initialize the system method manager (SMM).                            */

  smm_init();
  if (smm_error != AOK)
  {
    mm_error = MM_ERR_SYSTEM_METHOD_INIT;
    return;
  }

  /* Initialize the standard parameter input and output buffers.            */

  for (i = 0; i < MM_PBUFFER_SIZE; i++)
  {
    mm_stdin.buffer[i]  = 0;
    mm_stdout.buffer[i] = 0;
```

```
147     }
148     mm_stdin.index = mm_stdout.index = 0;
149     mm_stdin.bitindex = mm_stdout.bitindex = 0;
150
151     /* Initialize the system phone book data structure (NULL names, etc.).    */
152     /* Update (create) phone book with current MODBOT/unit information.       */
153     /* Reset mm_error if pb_init went OK (pb_init() affects mm_error).        */
154
155     pb_init();
156     if (pb_error != AOK)
157     {
158       mm_error = MM_ERR_IN_PHONEBOOK;
159       return;
160     }
161
162     mm_error = AOK;                         /* Function successful.           */
163   }
164
165
166   /*************************************************************************\
167   *                              mm_cycle                                   *
168   *************************************************************************
169   *                                                                        *
170   * Function:     Processes the next message (or next several messages) from *
171   *               the local communications interface. The parameter iterations *
172   *               specifies how many messages to process. Messages are taken *
173   *               from the LCI, decoded, and then processed accordingly.   *
174   *               Messages that are initiating action are passed to the local *
175   *               method manager (LMM), while all others are passed to the *
176   *               system method manager (SMM). Always cycles at least once. *
177   *               If the number of iterations is MM_CYCLE_FOREVER then this *
178   *               routine never returns (cycles continously processing local *
179   *               messages).                                               *
180   *                                                                        *
181   *               Also checks local and system trigger conditions for possible *
182   *               firing of methods, which may cause additional messages to be *
183   *               output.                                                  *
184   *                                                                        *
185   * Input:        mm_cycle(                                                *
186   *                 int iterations; number of messages to process.         *
187   *               );                                                       *
188   *                                                                        *
189   * Output:       Nothing.                                                 *
190   *                                                                        *
191   * Globals:      mm_error  : MM.C                                         *
192   *               lci_error : LCI.C                                        *
193   *                                                                        *
194   * Edit History: 12/20/90 - Written by Robin T. Laird.                    *
195   *                                                                        *
196   \*************************************************************************/
197
198   #define MM_SEND_ATTEMPTS        LCI_WAIT_FOREVER
199   #define MM_RECV_ATTEMPTS        8
200
201   void mm_cycle(iterations)
202   int iterations;
203   {
204     int          i, inc;
205     byte         status;
206     lci_message  l_in, l_out;
207     mm_message   m_in, m_out;
208
209     /* Repeat iterations times...                                           */
210     /* If iterations = MM_CYCLE_FOREVER then repeat forever (never return). */
211
212     if (iterations == MM_CYCLE_FOREVER)
213       inc = 0;
214     else
215       inc = 1;
216
217     /* Get a message from the local communications subsystem (if available). */
218     /* Decode message (network layer --> presentation layer).               */
219     /* If the message is initiating action pass to local method manager.    */
```



```
220     /* Else pass message to system method manager.                          */
221     /* Ignore errors on multiple iterations.                                */
222
223     i = 0;
224     do {
225       lci_receive_message(l_in, MM_RECV_ATTEMPTS);
226       if (lci_error != AOK)
227       {
228         mm_error = MM_ERR_MESSAGE_NOT_AVAILABLE;
229       }
230       else
231       {
232         mm_error = AOK;
233
234         mm_decode_message(l_in, &m_in);
235         switch(m_in.trans_disposition)
236         {
237           case MM_INITIATING:
238
239             /* Process the message as a local function activation.          */
240             /* Status variable indicates whether or not response msg needed. */
241
242             lmm_process(&m_in, &m_out, &status);
243             break;
244
245           case MM_COMMAND_RECEIVED:
246           case MM_COMMAND_EXECUTED:
247           case MM_COMMAND_UNKNOWN:
248           case MM_COMMAND_EXECUTION_FAILURE:
249
250             /* Process message as response to previously sent command.      */
251             /* Status variable indicates whether or not response msg needed. */
252
253             smm_process(&m_in, &m_out, &status);
254             break;
255
256           default:
257             mm_error = MM_ERR_MESSAGE_DISPOSITION;
258             continue;
259         }
260
261         /* If response required to above, encode message, send via LCI.     */
262
263         if (status == MM_RESPONSE_REQUIRED)
264         {
265           mm_encode_message(&m_out, l_out);
266           lci_send_message(l_out, MM_SEND_ATTEMPTS);
267         }
268       }
269
270       /* Check local and system trigger conditions.                         */
271       /* Fire appropriate methods. May cause messages to be sent.           */
272
273       lmm_fire();
274       smm_fire();
275
276     } while ((i+=inc) < iterations);
277   }
278
279
280   /*************************************************************************\
281   *                            mm_encode_message                           *
282   *************************************************************************
283   *                                                                        *
284   * Function:     Encodes the information contained in the mm_message      *
285   *               variable as a message of type lci_message. Each field of *
286   *               the input structure is encoded in the correct position in *
287   *               the output message as defined by the message format (below). *
288   *               The encoded message can then be transmitted to the local *
289   *               communications device (by the LCI subsystem).            *
290   *                                                                        *
291   *                       MM message in --> ENCODE --> LCI message out     *
292   *                                                                        *
```

```
293   *                The functions mm_decode_message() and mm_encode_message()    *
294   *                define the message format at this communications level.      *
295   *                Changes to this format must be reflected by changes to these *
296   *                two functions (AND ONLY THESE TWO FUNCTIONS).                *
297   *                                                                             *
298   * Input:         mm_encode_message(                                           *
299   *                  mm_message *to_encode; pointer to message to encode (in).  *
300   *                  lci_message encoded;   pointer to encoded message (out).   *
301   *                );                                                           *
302   *                                                                             *
303   * Output:        Nothing.                                                     *
304   *                                                                             *
305   * Globals:       mm_error : MM.C                                              *
306   *                                                                             *
307   * Edit History: 12/20/90 - Written by Robin T. Laird.                         *
308   *                                                                             *
309   \*****************************************************************************/
310
311   void mm_encode_message(to_encode, encoded)
312   mm_message *to_encode;
313   lci_message encoded;
314   {
315     byte i, j, l;
316
317     /* Inter-module message format (IMMF):                                     */
318
319     /*   DEST           SORC                                                 */
320     /*  DEVICE   MESG  DEVICE  SEQ   TRANSC    TRANSC   FN  PARAMS          */
321     /*    ID    LENGTH   ID    NUM  DISPOSIT  CATEGORY  ID  LENGTH  PARAMS  */
322     /*  ------------------------------------------------------------------   */
323     /*    8       8      8      8       4         4      8     8       8n    */
324
325     mm_error = AOK;                       /* Assume function successful.   */
326
327     i = MM_DEST_ADDR_POS;                 /* Index of first encoded byte.  */
328
329     encoded[i]   = to_encode->dest_modbot_id << MM_MODBOT_ID_BIT_SHIFT;
330     encoded[i++] |= to_encode->dest_unit_id & MM_UNIT_ID_MASK;
331
332     encoded[i++]  = to_encode->parameter_length + MM_COM_OVERHEAD_BYTES;
333
334     encoded[i]   = to_encode->src_modbot_id << MM_MODBOT_ID_BIT_SHIFT;
335     encoded[i++] |= to_encode->src_unit_id & MM_UNIT_ID_MASK;
336
337     encoded[i++]  = to_encode->sequence_number;
338
339     encoded[i]   = to_encode->trans_disposition << MM_TRANS_DISPOSIT_BIT_SHIFT;
340     encoded[i++] |= to_encode->trans_category & MM_TRANS_CATEGORY_MASK;
341
342     encoded[i++]  = to_encode->function_id;
343
344     encoded[i++]  = l = to_encode->parameter_length;
345     for (j = 0; j < l; j++) encoded[i++] = to_encode->parameter[j];
346   }
347
348
349   /*****************************************************************************\
350   *                            mm_decode_message                                *
351   *****************************************************************************
352   *                                                                             *
353   * Function:      Decodes the information contained in the mm_message          *
354   *                variable as a message of type lci_message. Each field of     *
355   *                the input structure is encoded in the correct position in    *
356   *                the output message as defined by the message format (below). *
357   *                                                                             *
358   *                        LCI message in --> DECODE --> MM message out         *
359   *                                                                             *
360   *                The functions mm_decode_message() and mm_encode_message()    *
361   *                define the message format at this communications level.      *
362   *                Changes to this format must be reflected by changes to these *
363   *                two functions (AND ONLY THESE TWO FUNCTIONS).                *
364   *                                                                             *
365   * Input:         mm_decode_message(                                           *
```



```
366   *                  lci_message to_decode; pointer to message to decode (in).  *
367   *                  mm_message *decoded;   pointer to message to encode (out). *
368   *                );                                                           *
369   *                                                                             *
370   * Output:        Nothing.                                                     *
371   *                                                                             *
372   * Globals:       mm_error : MM.C                                              *
373   *                                                                             *
374   * Edit History: 12/20/90 - Written by Robin T. Laird.                         *
375   *                                                                             *
376   \*****************************************************************************/
377
378   void mm_decode_message(to_decode, decoded)
379   lci_message to_decode;
380   mm_message *decoded;
381   {
382     byte i, j, l;
383
384     /* Inter-module message format (IMMF):                                     */
385
386     /*   DEST           SORC                                                 */
387     /*  DEVICE   MESG  DEVICE  SEQ   TRANSC    TRANSC   FN  PARAMS          */
388     /*    ID    LENGTH   ID    NUM  DISPOSIT  CATEGORY  ID  LENGTH  PARAMS  */
389     /*  ------------------------------------------------------------------   */
390     /*    8       8      8      8       4         4      8     8       8n    */
391
392     mm_error = AOK;                       /* Assume function successful.   */
393
394     i = MM_DEST_ADDR_POS;                 /* Index of first decoded byte.  */
395
396     decoded->dest_modbot_id    = to_decode[i] >> MM_MODBOT_ID_BIT_SHIFT;
397     decoded->dest_unit_id      = to_decode[i++] & MM_UNIT_ID_MASK;
398
399     decoded->message_length    = to_decode[i++];
400
401     decoded->src_modbot_id     = to_decode[i] >> MM_MODBOT_ID_BIT_SHIFT;
402     decoded->src_unit_id       = to_decode[i++] & MM_UNIT_ID_MASK;
403
404     decoded->sequence_number   = to_decode[i++];
405
406     decoded->trans_disposition = to_decode[i] >> MM_TRANS_DISPOSIT_BIT_SHIFT;
407     decoded->trans_category    = to_decode[i++] & MM_TRANS_CATEGORY_MASK;
408
409     decoded->function_id       = to_decode[i++];
410
411     decoded->parameter_length  = l = to_decode[i++];
412     for (j = 0; j < l; j++) decoded->parameter[j] = to_decode[i++];
413   }
414
415
416   /*****************************************************************************\
417   *                               mm_sprintfb                                   *
418   *****************************************************************************
419   *                                                                             *
420   * Function:      Provides formatted output of binary data as in sprintf().    *
421   *                Used to place data into the parameter passing portion of     *
422   *                a message (i.e., the end of the message). Variables are      *
423   *                output according to special % flags as in printf() but       *
424   *                as binary values NOT ASCII. A maximum number of values can   *
425   *                be passed as parameters and is compiler dependent. The only  *
426   *                "flags" that are supported indicate the types and length of  *
427   *                parameters as in %10s. The format is similar to sprintf().   *
428   *                                                                             *
429   *                Only the type flags are currently supported.                 *
430   *                                                                             *
431   *                Care must be taken to set/reset the buffer indexes between   *
432   *                function calls. Otherwise the buffer may overflow...         *
433   *                                                                             *
434   *                The types currently supported are:                           *
435   *                                                                             *
436   *                Character       Type            Output Length                *
437   *                ------------------------------------------------------       *
438   *                  %y            byte            1  bit                       *
```

```
439   *                   %b          byte            8  bits                *
440   *                   %c          char            8  bits                *
441   *                   %u          unsigned int    16 bits                *
442   *                   %d          int             16 bits                *
443   *                   %l          long int        32 bits                *
444   *                   %s          pointer         8n bits                *
445   *                                                                      *
446   * Input:          mm_sprintfb(                                         *
447   *                   mm_pbuffer; pointer to output buffer (where data goes).  *
448   *                   char *format; pointer to data output format string.  *
449   *                   ...;        data values to be output.              *
450   *                 );                                                   *
451   *                                                                      *
452   * Output:         Nothing.                                             *
453   *                                                                      *
454   * Globals:        mm_error : MM.C                                      *
455   *                                                                      *
456   * Edit History: 12/20/90 - Written by Robin T. Laird.                  *
457   *                                                                      *
458   \**********************************************************************/
459
460   void mm_sprintfb(p, format, a1, a2, a3)
461   mm_pbuffer *p;
462   char *format;
463   double a1, a2, a3;
464   {
465     va_list ap;
466     char    *c, *sval;
467     int     ival;
468     long    lval;
469
470     /* Initialize the variable argument macro pointers.                 */
471
472     va_start(ap, format);
473
474     /* Loop through the format string, process according to control chars.  */
475     /* Access arguments using va_arg(), inc index and bit index accordingly.  */
476     /* Index and bitindex indicate number of bytes and bits in last byte.  */
477
478     for (c = format; *c; c++)
479     {
480       if (*c != '%')
481       {
482         if (p->bitindex)
483         {
484           p->index++;
485           p->bitindex = 0;
486         }
487         p->buffer[p->index++] = *c;
488         continue;
489       }
490
491       switch(*++c)
492       {
493         case 'y': /* bit */
494           if (p->bitindex == 0) p->buffer[p->index] = 0;
495
496           /* Franklin and Microsoft differ in way bytes are managed.    */
497
498           #if defined(MSDOS)
499           p->buffer[p->index] |= (va_arg(ap, int) << p->bitindex);
500           #else
501           p->buffer[p->index] |= (va_arg(ap, byte) << p->bitindex);
502           #endif
503
504           if (++p->bitindex == 8)
505           {
506             p->bitindex = 0;
507             p->index++;
508           }
509           break;
510
511         case 'b': /* byte */
```



```
512         case 'c': /* char */
513           if (p->bitindex)
514           {
515             p->index++;
516             p->bitindex = 0;
517           }
518
519           /* Franklin and Microsoft differ in way bytes are managed.    */
520
521           #if defined(MSDOS)
522           p->buffer[p->index++] = va_arg(ap, int);
523           #else
524           p->buffer[p->index++] = va_arg(ap, byte);
525           #endif
526
527           break;
528
529         case 'u': /* unsigned int */
530         case 'd': /* int */
531           if (p->bitindex)
532           {
533             p->index++;
534             p->bitindex = 0;
535           }
536           ival = va_arg(ap, int);
537           p->buffer[p->index++] = ival >> 8;
538           p->buffer[p->index++] = ival & 0x00FF;
539           break;
540
541         case 'l': /* long */
542           if (p->bitindex)
543           {
544             p->index++;
545             p->bitindex = 0;
546           }
547           lval = va_arg(ap, long);
548           p->buffer[p->index++] = lval >> 24;
549           p->buffer[p->index++] = lval >> 16;
550           p->buffer[p->index++] = lval >>  8;
551           p->buffer[p->index++] = lval & 0x000000FFL;
552           break;
553
554         case 's': /* pointer (string) */
555           if (p->bitindex)
556           {
557             p->index++;
558             p->bitindex = 0;
559           }
560           sval = va_arg(ap, char*);
561           do {
562             p->buffer[p->index++] = (byte)*sval;
563           } while (*sval++);
564           break;
565
566         default:
567           if (p->bitindex)
568           {
569             p->index++;
570             p->bitindex = 0;
571           }
572           p->buffer[p->index++] = *c;
573       }
574     }
575
576     /* Clean up after moving argument pointer.                          */
577
578     va_end(ap);
579   }
580
581
582   /**********************************************************************\
583   *                                mm_sscanfb                            *
584   **********************************************************************
```

```
585   *                                                                       *
586   * Function:     Provides formatted input of binary data as in sscanf().   *
587   *               Used to extract data from the parameter passing portion of *
588   *               a message (i.e., the end of the message). Variables are    *
589   *               input according to special % flags as in printf() but      *
590   *               as binary values NOT ASCII. A maximum number of values can *
591   *               be passed as parameters and is compiler dependent. The only *
592   *               "flags" that are supported indicate the types and length of *
593   *               parameters as in %l0s. The format is similar to sscanf().  *
594   *                                                                          *
595   *               Only the type flags are currently supported.               *
596   *                                                                          *
597   *               Care must be taken to set/reset the buffer indexes between *
598   *               function calls. Otherwise the buffer may overflow...       *
599   *                                                                          *
600   *               The types currently supported are:                         *
601   *                                                                          *
602   *               Character       Type            Input Length               *
603   *               ------------------------------------------------------     *
604   *                  %y           byte            1  bit                     *
605   *                  %b           byte            8  bits                    *
606   *                  %c           char            8  bits                    *
607   *                  %u           unsigned int    16 bits                    *
608   *                  %d           int             16 bits                    *
609   *                  %l           long int        32 bits                    *
610   *                  %s           pointer         8n bits                    *
611   *                                                                          *
612   * Input:        mm_sscanf(                                                 *
613   *                 mm_pbuffer;   pointer to input buffer (source of data).  *
614   *                 char *format; pointer to data input format string.       *
615   *                 ...;          data values to be output.                  *
616   *               );                                                         *
617   *                                                                          *
618   * Output:       Nothing.                                                   *
619   *                                                                          *
620   * Globals:      mm_error : MM.C                                            *
621   *                                                                          *
622   * Edit History: 12/20/90 - Written by Robin T. Laird.                      *
623   *                                                                          *
624   \**************************************************************************/
625
626   void mm_sscanf( .. format, a1, a2, a3)
627   mm_pbuffer *p;
628   char *format;
629   double a1, a2, a3;
630   {
631     va_list ap;
632     char    *c, *sval;
633     int     ival;
634     long    lval;
635
636     /* Initialize the variable argument macro pointers.                    */
637
638     va_start(ap, format);
639
640     /* Loop through the format string, process according to control chars.  */
641     /* Access arguments using va_arg(), inc index and bit index accordingly. */
642     /* Index and bitindex indicate number of bytes and bits in last byte.  */
643
644     for (c = format; *c; c++)
645     {
646       if (*c != '%')
647       {
648         continue;
649       }
650
651       switch(*++c)
652       {
653         case 'y': /* bit */
654           *(va_arg(ap, byte*)) = (p->buffer[p->index] >> p->bitindex) & 0x01;
655           if (++p->bitindex == 8)
656           {
657             p->bitindex = 0;
```



```
658             p->index++;
659           }
660           break;
661
662         case 'b': /* byte */
663         case 'c': /* char */
664           if (p->bitindex)
665           {
666             p->index++;
667             p->bitindex = 0;
668           }
669           *(va_arg(ap, byte*)) = p->buffer[p->index++];
670           break;
671
672         case 'u': /* unsigned int */
673         case 'd': /* int */
674           if (p->bitindex)
675           {
676             p->index++;
677             p->bitindex = 0;
678           }
679           ival  = (int)p->buffer[p->index++] << 8;
680           ival |= (int)p->buffer[p->index++];
681           *(va_arg(ap, int*)) = ival;
682           break;
683
684         case 'l': /* long */
685           if (p->bitindex)
686           {
687             p->index++;
688             p->bitindex = 0;
689           }
690           lval  = (long)p->buffer[p->index++] << 24;
691           lval |= (long)p->buffer[p->index++] << 16;
692           lval |= (long)p->buffer[p->index++] <<  8;
693           lval |= (long)p->buffer[p->index++];
694           *(va_arg(ap, long*)) = lval;
695           break;
696
697         case 's': /* pointer (string) */
698           if (p->bitindex)
699           {
700             p->index++;
701             p->bitindex = 0;
702           }
703           sval = va_arg(ap, char*);
704           do {
705             *sval++ = (char)p->buffer[p->index];
706           } while (p->buffer[p->index++]);
707           break;
708
709         default:
710           break;
711       }
712     }
713
714     /* Clean up after moving argument pointer.                             */
715
716     va_end(ap);
717   }
718
719
720   /***************************************************************************\
721   *                              mm_service_event                             *
722   ****************************************************************************
723   *                                                                          *
724   * Function:     Removes the completed event from the system method queue,  *
725   *               and returns the response message in the output parameter   *
726   *               message (m_out). The parameter portion of the message is   *
727   *               also copied to the standard input I/O buffer (mm_stdin).   *
728   *               Periodic events are NOT removed from the queue, but the    *
729   *               information portion of the event is still returned.        *
730   *                                                                          *
```

```
731   * Input:         mm_service_event(                                        *
732   *                   Int event;         number of event to service.       *
733   *                   mm_message *m_out; pointer to message holding response.*
734   *                 );                                                     *
735   *                                                                        *
736   * Output:        Nothing.                                                 *
737   *                                                                        *
738   * Globals:       smm_error : module SMM.C                                 *
739   *                item      : module SMM.C                                 *
740   *                queue     : module SMM.C (side effect)                   *
741   *                mm_error  : module MM.C                                  *
742   *                mm_stdin  : module MM.C                                  *
743   *                                                                        *
744   * Edit History: 01/24/91 - Written by Robin T. Laird.                    *
745   *                                                                        *
746   \************************************************************************/
747
748   void mm_service_event(event, m_out)
749   int event;
750   mm_message *m_out;
751   {
752     mm_error = AOK;                          /* Assume function successful.      */
753
754     /* Remove event from method queue if event is NOT periodic.               */
755     /* Event is actually acting as a node pointer.                            */
756     /* The event may not be at beginning of the queue.                        */
757     /* Note that the parameter event is NOT checked for validity.             */
758
759     if (item[event].info.trans_category == MM_PERIODIC_STATUS_REQUEST)
760     {
761       *m_out = item[event].info;
762
763       /* Reset trans disposition to indicate no new message.                  */
764
765       item[event].info.trans_disposition = MM_AWAITING_EVENT;
766     }
767     else
768     {
769       if (event == queue)
770         smm_remove_q(&queue, m_out);
771       else
772         smm_remove_q(&event, m_out);
773     }
774
775     /* Copy parameter information from message to standard input buffer.      */
776
777     memcpy(mm_stdin.buffer, m_out->parameter, (int)m_out->parameter_length);
778     mm_stdin.index    = 0;
779     mm_stdin.bitindex = 0;
780   }
781
782
783   /*************************************************************************\
784   *                            mm_terminate_event                           *
785   *************************************************************************
786   *                                                                        *
787   * Function:      Removes the completed event from the system method queue.*
788   *                Periodic events are removed from the queue.             *
789   *                A message is sent to the event handler of the appropriate*
790   *                module to indicate that the specified periodic event should*
791   *                be terminated (by setting its period to 0).             *
792   *                                                                        *
793   * Input:         mm_terminate_event(                                      *
794   *                  Int event; number of event to terminate.              *
795   *                );                                                      *
796   *                                                                        *
797   * Output:        Nothing.                                                 *
798   *                                                                        *
799   * Globals:       smm_error : module SMM.C                                 *
800   *                queue     : module SMM.C (side effect)                   *
801   *                mm_error  : module MM.C                                  *
802   *                                                                        *
803   * Edit History: 03/28/91 - Written by Robin T. Laird.                    *
```



```
804   *                                                                        *
805   \************************************************************************/
806
807   #define MM_TERM_ATTEMPTS                LCI_WAIT_FOREVER
808
809   void mm_terminate_event(event)
810   int event;
811   {
812     lci_message l_out;
813     mm_message  m_out;
814
815     mm_error = AOK;                        /* Assume function successful.      */
816
817     /* Remove event from method queue.                                        */
818     /* Event is actually acting as a node pointer.                            */
819     /* The event may not be at beginning of the queue.                        */
820     /* Note that the parameter event is NOT checked for validity.             */
821
822     if (event == queue)
823       smm_remove_q(&queue, &m_out);
824     else
825       smm_remove_q(&event, &m_out);
826
827     /* Change message length accordingly (overhead plus period length).       */
828     /* Change transaction disposition to indicate initiating command.         */
829     /* Set parameter field to indicate zero (0) period.                       */
830
831     m_out.message_length       = MM_COM_OVERHEAD_BYTES + MM_PLEN;
832     m_out.trans_disposition    = MM_INITIATING;
833     m_out.parameter_length     = MM_PLEN;
834     m_out.parameter[MM_PPOS]   = 0;
835     m_out.parameter[MM_PPOS+1] = 0;
836
837     /* Encode the above information for transmission via the LCI.             */
838     /* And send it...                                                         */
839
840     mm_encode_message(&m_out, l_out);
841     lci_send_message(l_out, MM_TERM_ATTEMPTS);
842   }
843
844
845   /*************************************************************************\
846   *                              mm_check_event                            *
847   *************************************************************************
848   *                                                                        *
849   * Function:      Returns the status of the parameter event. If the event *
850   *                has completed, then the transaction disposition is returned,*
851   *                indicating the success of the method activation associated*
852   *                with the event. If the event has not completed, then the *
853   *                value MM_AWAITING_EVENT is returned. If the queue is empty*
854   *                then NULL is returned.                                  *
855   *                                                                        *
856   * Input:         mm_check_event(                                          *
857   *                  Int event; number of event to check.                  *
858   *                );                                                      *
859   *                                                                        *
860   * Output:        Disposition of completed event or MM_AWAITING_EVENT.     *
861   *                                                                        *
862   * Globals:       queue     : module SMM.C                                 *
863   *                item      : module SMM.C                                 *
864   *                mm_error  : module MM.C                                  *
865   *                                                                        *
866   * Edit History: 01/24/91 - Written by Robin T. Laird.                    *
867   *                                                                        *
868   \************************************************************************/
869
870   int mm_check_event(event)
871   int event;
872   {
873     /* Make sure the queue is not empty (report error if it is).              */
874     /* Then report AOK and return transaction disposition.                    */
875     /* Note that the parameter event is NOT checked for validity.             */
876
```

```
877     if (smm_empty_q(queue))
878     {
879       mm_error = MM_ERR_NO_EVENT_PENDING;
880       return(NULL);
881     }
882     else
883     {
884       mm_error = AOK;
885       return((int)item[event].info.trans_disposition);
886     }
887   }
```

```
1   /*****************************************************************************\
2   *                                   PB.H                                      *
3   *****************************************************************************
4   *                                                                             *
5   * CPCI:          IED90-MRA-COM-MMS-PB-H-R0C0                                  *
6   *                                                                             *
7   * Description:   Method Manager Phone Book variables and functions.           *
8   *                Contains constant function parameter declarations (#defines) *
9   *                as well as function return values (for success and failure   *
10  *                of all operations). Contains the function prototypes for the *
11  *                PB.C module.                                                 *
12  *                                                                             *
13  *                Module PB exports the following types/variables/functions:   *
14  *                                                                             *
15  *                int pb_error;                                                *
16  *                                                                             *
17  *                pb_init();                                                   *
18  *                pb_update_pb();                                              *
19  *                pb_lookup_pb();                                              *
20  *                                                                             *
21  * Notes:         1) See SDS pp. 5-6 through 5-x for more information.         *
22  *                                                                             *
23  * Edit History: 02/04/91 - Written by Robin T. Laird.                         *
24  *                                                                             *
25  \*****************************************************************************/
26
27  #ifndef PB_MODULE_CODE
28  #define PB_MODULE_CODE                  11000
29
30  /* Public Data Structures:                                                    */
31
32  #define PB_ERR_NOT_INIT                 1+PB_MODULE_CODE
33  #define PB_ERR_ADDING_EVENT             2+PB_MODULE_CODE
34  #define PB_ERR_UNIT_NOT_FOUND           3+PB_MODULE_CODE
35
36  /* External module global error variable.                                    */
37
38  extern int pb_error;
39
40  /* Public Functions:                                                          */
41
42  void pb_init();
43  void pb_update(char *name);
44  int  pb_lookup(char *name, byte *modbot_addr, byte *unit_addr);
45
46  #endif
```

```
1   /*****************************************************************************\
2   *                                  PB.C                                       *
3   *****************************************************************************
4   *                                                                            *
5   * CPCI:          JED90-MRA-COM-MMS-PB-C-R0C0                                 *
6   *                                                                            *
7   * Description:   Phone Book manager functions.                               *
8   *                Implements the Method Manager Phone Book manager module.    *
9   *                This module contains the functions that create and update   *
10  *                the system Phone Book which contains the names and addresses *
11  *                of all units within the MODBOT system (along the MODBUS     *
12  *                copbunications network).                                    *
13  *                                                                            *
14  *                Module PB exports the following variables/functions:        *
15  *                                                                            *
16  *                int pb_error;                                               *
17  *                                                                            *
18  *                pb_init();                                                  *
19  *                pb_update_pb();                                             *
20  *                pb_lookup_pb();                                             *
21  *                                                                            *
22  * Notes:         1) The PB functions are implementation independent.         *
23  *                                                                            *
24  * Edit History: 02/04/91 - Written by Robin T. Laird.                        *
25  *                                                                            *
26  \*****************************************************************************/
27
28  #include <string.h>                     /* Standard string functions.       */
29  #include <sysdefs.h>                    /* System constants and types.      */
30  #include <rtc.h>                        /* Real-time clock functions.       */
31  #include "mm.h"                         /* Method manager.                  */
32  #include "snm.h"                        /* System ethod manager.            */
33  #include "pb.h"                         /* System method manager.           */
34
35  #if defined(DEBUG)
36  #include <debug.h>
37  #endif
38
39  /* Public Variables:                                                         */
40
41  /* Global module error variable, pb_error.                                  */
42  /* pb_error contains code of last error occurence.                           */
43  /* Should be set to AOK after each successful function call.                 */
44  /* Variable can be examined by other software after each function call.      */
45
46  XDATA int pb_error = PB_ERR_NOT_INIT;   /* Global module error variable.     */
47
48  /* The system phone book consists of listings for each unit on each MODBOT.  */
49  /* Each MODBOT has an associated address and a number of unit entries.       */
50  /* Each phone book entry has a unit name and a unit address.                 */
51  /* The phone book is sorted by unit name at system start-up (pb_update()).   */
52
53  #define MAX_NAME_LEN            8
54  #define MAX_UNITS               32
55  #define MAX_MODBOTS             1
56
57  typedef struct { char name[MAX_NAME_LEN];
58                   byte unit_addr;
59                 } unit_entry;
60
61  typedef struct { unit_entry unit[MAX_UNITS];
62                   byte       modbot_addr;
63                   byte       num_entries;
64                 } modbot_entry;
65
66  static XDATA modbot_entry pb[MAX_MODBOTS];
67
68
69  /*****************************************************************************\
70  *                                pb_init                                     *
71  *****************************************************************************
72  *                                                                            *
73  * Function:      Initializes the Method Manager Phone Book data structure.   *
```



```
74  *                Number of phone book entries for each MODBOT is set to zero. *
75  *                Names and addresses are set to NULL.                        *
76  *                                                                            *
77  * Input:         pb_init();                                                  *
78  *                                                                            *
79  * Output:        Nothing.                                                    *
80  *                                                                            *
81  * Globals:       pb_error  : PB.C                                            *
82  *                pb        : PB.C                                            *
83  *                                                                            *
84  * Edit History: 12/20/90 - Written by Robin T. Laird.                        *
85  *                                                                            *
86  \*****************************************************************************/
87
88  void pb_init()
89  {
90    byte modbot, unit;
91
92    pb_error = AOK;                       /* Assume function successful.     */
93
94    /* Initialize the system phone book data structure (NULL names, etc.).     */
95    /* Update (create) phone book with current MODBOT/unit information.        */
96
97    for (modbot = 0; modbot < MAX_MODBOTS; modbot++)
98    {
99      pb[modbot].modbot_addr = 0;
100     pb[modbot].num_entries = 0;
101     for (unit = 0; unit < MAX_UNITS; unit++)
102     {
103       pb[modbot].unit[unit].name[0]   = '\0';
104       pb[modbot].unit[unit].unit_addr = 0;
105     }
106   }
107   pb_update(NULLPTR);
108 }
109
110
111 /*****************************************************************************\
112 *                                pb_update                                   *
113 *****************************************************************************
114 *                                                                            *
115 * Function:      Creates the system phone book (or updates the entry for     *
116 *                the unit that corresponds to the parameter name). If the    *
117 *                parameter name is NULLPTR, then each MODBOT is queried for  *
118 *                available units connected to its local network. Units       *
119 *                that are capable of responding are added to the phone book  *
120 *                along with their addresses. The function also updates an    *
121 *                entry for a given name (the same procedure is followed      *
122 *                except the function halts when the named unit is found      *
123 *                or when the maximum number of units has been searched).     *
124 *                                                                            *
125 * Input:         pb_update(                                                  *
126 *                  char *name; pointer to name of unit to update (or NULLPTR).*
127 *                );                                                          *
128 *                                                                            *
129 * Output:        Nothing.                                                    *
130 *                                                                            *
131 * Globals:       pb_error : PB.C                                             *
132 *                pb       : PB.C                                             *
133 *                                                                            *
134 * Edit History: 01/02/91 - Written by Robin T. Laird.                        *
135 *                                                                            *
136 \*****************************************************************************/
137
138 #define ONE_TIME                1       /* Cycle method manager once.       */
139 #define WAIT_TIME               1500    /* Time in ms to wait for response. */
140
141 void pb_update(name)
142 char *name;
143 {
144   int           event, status;
145   byte          modbot, unit;
146   mm_message    m_in;
```

```
147     unsigned long stop_at;
148
149     pb_error = AOK;                         /* Assume function successful.      */
150
151     /* Query each (even numbered) unit on each MODBOT (could take a while).   */
152     /* Send "QUERY NAME" message to unit at current MODBOT/unit address.      */
153     /* If the unit responds then add name to phone book at current address.   */
154     /* If we're looking for a particular name, then stop when (if) found.     */
155     /* Variable no_change indicates if anything changed in the phone book.    */
156
157     for (modbot = 0; modbot < MAX_MODBOTS; modbot++)
158       for (unit = 0; unit < MAX_UNITS; unit+=2)
159       {
160         if ((event = unit_name(modbot, unit)) == MM_NULL_EVENT)
161         {
162           pb_error = PB_ERR_ADDING_EVENT;
163         }
164         else
165         {
166           /* Check for event completion. Timeout period should be short.      */
167
168           stop_at = rtc_time() + WAIT_TIME;
169           do {
170             mm_cycle(ONE_TIME);
171             status = mm_check_event(event);
172           } while (rtc_time() < stop_at && status == MM_AWAITING_EVENT);
173           mm_service_event(event, &m_in);
174
175           /* If event completed OK, then add entry to phonebook.              */
176
177           if (status != MM_AWAITING_EVENT && mm_error == AOK)
178           {
179             mm_sscanfb(&mm_stdin, "%s", pb[modbot].unit[unit].name);
180             pb[modbot].modbot_addr          = modbot;
181             pb[modbot].unit[unit].unit_addr = unit;
182             pb[modbot].num_entries++;
183
184             /* If we're updating a particular unit, stop here if name match.  */
185
186             if (name != NULLPTR && strcmp(name, pb[modbot].unit[unit].name) == 0)
187             {
188               pb_error = AOK;
189               return;
190             }
191           }
192           else
193           {
194             /* Delete (NULL) this entry since unit didn't respond.            */
195
196             pb[modbot].unit[unit].name[0]   = '\0';
197             pb[modbot].unit[unit].unit_addr = 0;
198           }
199         }
200       }
201   }
202
203
204   /*************************************************************************\
205   *                              pb_lookup                                  *
206   *************************************************************************
207   *                                                                        *
208   * Function:     Looks up the parameter unit name in the sytem phonebook. *
209   *               If the entry is found, then the corresponding MODBOT/unit*
210   *               address is returned. Otherwise, the global error variable*
211   *               pb_error is set to PB_ERR_UNIT_NOT_FOUND. A binary search *
212   *               method is used to locate the name in the phonebook.      *
213   *                                                                        *
214   * Input:        pb_lookup(                                               *
215   *                 char *name;        pointer to name of unit to look up. *
216   *                 byte *modbot_addr; pointer to returned MODBOT address. *
217   *                 byte *unit_addr;   pointer to returned unit address.   *
218   *               );                                                       *
219   *                                                                        *
```



```
220   * Output:       Integer indicating if the named unit was found (TRUE/FALSE). *
221   *                                                                        *
222   * Globals:      pb_error : PB.C                                          *
223   *               pb       : PB.C                                          *
224   *                                                                        *
225   * Edit History: 01/22/91 - Written by Robin T. Laird.                    *
226   *                                                                        *
227   \************************************************************************/
228
229   int pb_lookup(name, modbot_addr, unit_addr)
230   char *name;
231   byte *modbot_addr, *unit_addr;
232   {
233     byte modbot, unit;
234     byte low, high, mid, cond;
235
236     pb_error = AOK;                         /* Assume function successful.      */
237
238     /* Search each unit on each MODBOT. Linear search is used.                */
239
240     for (modbot = 0; modbot < MAX_MODBOTS; modbot++)
241     {
242       for (unit = 0; unit < MAX_UNITS; unit+=2)
243       {
244         if (strcmp(name, pb[modbot].unit[unit].name) == 0)
245         {
246           *modbot_addr = pb[modbot].modbot_addr;
247           *unit_addr   = pb[modbot].unit[unit].unit_addr;
248           return(TRUE);
249         }
250       }
251     }
252
253     /* Entry not found. Set error variable and return 0 for index.           */
254
255     pb_error = PB_ERR_UNIT_NOT_FOUND;
256     return(FALSE);
257   }
```

```
/*****************************************************************************\
*                                  SMM.H                                      *
*****************************************************************************
*                                                                            *
* CPCI:          JED90-MRA-COM-MMS-SMM-H-ROC1                                *
*                                                                            *
* Description:   System Method Manager (SMM) variables and functions.        *
*                Contains constant function parameter declarations (#defines)*
*                as well as function return values (for success and failure  *
*                of all operations). Contains the function prototypes for the*
*                SMM.C module.                                               *
*                                                                            *
*                Defines the library functions for the inherited classes.    *
*                Each .LIB file specifies the functions available from the   *
*                particular MODBOT unit. Collectively, these functions       *
*                represent all of the external functions (A.K.A., methods)   *
*                that an application anticipates using.                      *
*                                                                            *
*                Module SMM exports the following variables/functions:       *
*                                                                            *
*                int smm_error;                                              *
*                                                                            *
*                smm_init();                                                 *
*                smm_process();                                              *
*                smm_fire();                                                 *
*                smm_generate_message();                                     *
*                smm_translate_message();                                    *
*                                                                            *
*                OBJECT (STATUS_REQUEST):                                    *
*                obj_class();                                                *
*                obj_superclass();                                           *
*                                                                            *
*                UNIT (STATUS_REQUEST):                                      *
*                unit_name();                                                *
*                                                                            *
*                UNIT (CONTROL, CONTROL_WITH_ACK):                           *
*                unit_reset();                                               *
*                                                                            *
* Notes:         1) See SDS pp. 5-6 through 5-x for more information.        *
*                                                                            *
* Edit History: 10/01/90 - Written by Robin T. Laird.                        *
*                                                                            *
\*****************************************************************************/

#ifndef SMM_MODULE_CODE
#define SMM_MODULE_CODE                 10000

/* Public Data Structures:                                                   */

#define SMM_ERR_NOT_INIT                1+SMM_MODULE_CODE
#define SMM_ERR_MSG_TRANSLATION         2+SMM_MODULE_CODE
#define SMM_ERR_MSG_GENERATION          3+SMM_MODULE_CODE
#define SMM_ERR_NO_EVENT_MATCH          4+SMM_MODULE_CODE

#define SMM_ERR_EMPTY_QUEUE             5+SMM_MODULE_CODE
#define SMM_ERR_FULL_QUEUE              6+SMM_MODULE_CODE

/* External module global error variable.                                   */

extern int smm_error;

/* Public Functions:                                                         */

void smm_init();
void smm_process(mm_message *m_in, mm_message *m_out, byte *status);
void smm_fire(void);
void smm_generate_message(char *dest_unit_name, mm_message *m_in, int *event);
void smm_translate_message(mm_message *m_in, mm_message *m_out, byte *status);

/* Inherited Public Functions:                                               */

int obj_class(byte modbot, byte unit);
int obj_superclass(byte modbot, byte unit);
```



```
int unit_name(byte modbot, byte unit);
int unit_reset(byte modbot, byte unit);

/* Number of inherited functions (A.K.A., methods) is given below.          */
/* Includes object and unit class methods defined herein.                   */

#define SMM_NUM_INHERITED_FNS   4

/* Function identification numbers for object class methods.                */

#define OBJ_FN_CLASS            0
#define OBJ_FN_SUPERCLASS       1

/* Function identification numbers for unit class methods.                  */

#define UNIT_FN_NAME            2
#define UNIT_FN_RESET           3

#endif
```

```
1   /*****************************************************************************\
2   *                                  SMM.C                                      *
3   ******************************************************************************
4   *                                                                            *
5   * CPCI:          IED90-MRA-COM-MMS-SMM-C-R0C1                                *
6   *                                                                            *
7   * Description:   MMS system method manager functions.                        *
8   *                Implements the MRA standard System Method Manager (SMM)     *
9   *                module. This module contains the functions required to      *
10  *                process incoming messages that represent system method      *
11  *                activation requests from external processes. It manages     *
12  *                the MPU method activation queue data structure that manages *
13  *                execution and response of MODBOT unit functions.            *
14  *                                                                            *
15  *                Module SMM exports the following variables/functions:       *
16  *                                                                            *
17  *                int smm_error;                                              *
18  *                                                                            *
19  *                smm_init();                                                 *
20  *                smm_process();                                              *
21  *                smm_fire();                                                 *
22  *                smm_generate_message();                                     *
23  *                smm_translate_message();                                    *
24  *                                                                            *
25  * Notes:         1) The SMM functions are implementation independent.        *
26  *                3) This module is NOT a stand-alone compilation unit.       *
27  *                   It is included by the module MM.C and is compiled there. *
28  *                   It is assumed that the file SMM.H is included before it. *
29  *                                                                            *
30  * Edit History: 12/20/90 - Written by Robin T. Laird.                        *
31  *                                                                            *
32  *****************************************************************************/
33
34  /* Public Variables:                                                        */
35
36  /* Global module error variable, smm_error.                                */
37  /* smm_error contains code of last error occurence.                         */
38  /* Should be set to AOK after each successful function call.                */
39  /* Variable can be examined by other software after each function call.     */
40
41  XDATA int smm_error = SMM_ERR_NOT_INIT; /* Global module error variable.    */
42
43  /* Local error values and list-specific literals.                           */
44
45  #define SMM_ERR_OVERFLOW                7+SMM_MODULE_CODE
46  #define SMM_ERR_INVALID_PTR             8+SMM_MODULE_CODE
47
48  #define SMM_NULL_PTR                    -1
49  #define SMM_MAX_NODES                   16
50
51  /* Node pointers for array-based queues are simply integer indexes.         */
52
53  typedef int nodeptr;
54
55  /* Node for doubly-linked queue has left, right pointers along with data.   */
56  /* The event identifiers match the node index (in item below) with the node.*/
57  /* So, for example, the event identifier for item[4] is 4.                  */
58
59  typedef struct { mm_message info;
60                   int event;
61                   nodeptr left;
62                   nodeptr right;
63                 } node;
64
65  /* Node "pool" is an array of uninitialized nodes.                          */
66
67  static XDATA node item[SMM_MAX_NODES];
68
69  /* Avail queue is initialized to hold all nodes.                            */
70  /* Only right pointers are maintained on avail list.                        */
71  /* Global queue contains events that require processing.                    */
72
73  static XDATA nodeptr avail, queue;
```



```
74
75  /* System method manager trigger condition table.                           */
76  /* Holds method activation conditions and corresponding method to activate. */
77  /* Var smm_num_methods holds count of methods in trigger condition table.   */
78
79  #define SMM_MAX_METHODS         1       /* Number of methods we can hold.   */
80
81  typedef struct { mm_message    method;
82                   word          period;
83                   unsigned long fireat;
84                 } smm_cond;
85
86  static XDATA int smm_num_methods = 0;
87  static XDATA smm_cond smm_trigger[SMM_MAX_METHODS];
88
89
90  /*****************************************************************************\
91  *                                smm_full_q                                   *
92  ******************************************************************************
93  *                                                                            *
94  * Function:      Function that returns the boolean of whether or not the     *
95  *                parameter queue is full (TRUE if so, FALSE if not).         *
96  *                                                                            *
97  * Input:         smm_full_q(                                                 *
98  *                  nodeptr p; pointer to (index of) beginning of queue.      *
99  *                );                                                          *
100 *                                                                            *
101 * Output:        Integer, TRUE if queue FULL, FALSE if queue not FULL.       *
102 *                                                                            *
103 * Globals:       avail : module SMM.C                                        *
104 *                                                                            *
105 * Edit History: 07/08/90 - Written by Robin T. Laird.                        *
106 *                                                                            *
107 \*****************************************************************************/
108
109 static int smm_full_q(p)
110 nodeptr p;
111 {
112   return (avail == SMM_NULL_PTR);
113 }
114
115
116 /*****************************************************************************\
117 *                                smm_empty_q                                 *
118 ******************************************************************************
119 *                                                                            *
120 * Function:      Function that returns the boolean of whether or not the     *
121 *                parameter queue is empty (TRUE if so, FALSE if not).        *
122 *                                                                            *
123 * Input:         smm_empty_q(                                                *
124 *                  nodeptr p; pointer to (index of) beginning of queue.      *
125 *                );                                                          *
126 *                                                                            *
127 * Output:        Integer, TRUE if queue EMPTY, FALSE if queue not EMPTY.     *
128 *                                                                            *
129 * Globals:       None.                                                       *
130 *                                                                            *
131 * Edit History: 07/08/90 - Written by Robin T. Laird.                        *
132 *                                                                            *
133 \*****************************************************************************/
134
135 static int smm_empty_q(p)
136 nodeptr p;
137 {
138   return (p == SMM_NULL_PTR);
139 }
140
141
142 /*****************************************************************************\
143 *                                smm_avail_init                              *
144 ******************************************************************************
145 *                                                                            *
146 * Function:      Initializes the global available node list. The avail list   *
```

```
147   *                is a singly-linked list that contains nodes available for   *
148   *                use on other list structures (like the global event queue).  *
149   *                                                                             *
150   * Input:         smm_avail_init(                                              *
151   *                  nodeptr *p; pointer to beginning of avail list.            *
152   *                );                                                           *
153   *                                                                             *
154   * Output:        Nothing.                                                     *
155   *                                                                             *
156   * Globals:       smm_error : module SMM.C                                     *
157   *                item      : module SMM.C                                     *
158   *                                                                             *
159   * Edit History: 02/15/91 - Written by Robin T. Laird.                         *
160   *                                                                             *
161   \*****************************************************************************/
162
163   static void smm_avail_init(p)
164   nodeptr *p;
165   {
166     nodeptr q;
167
168     smm_error = AOK;
169
170     for (q = 0; q < SMM_MAX_NODES-1; q++)
171     {
172       item[q].event = q;
173       item[q].left  = SMM_NULL_PTR;
174       item[q].right = q+1;
175     }
176     item[SMM_MAX_NODES-1].event = q;
177     item[SMM_MAX_NODES-1].left  = SMM_NULL_PTR;
178     item[SMM_MAX_NODES-1].right = SMM_NULL_PTR;
179
180     *p = 0;
181   }
182
183
184   /*****************************************************************************\
185   *                                  smm_get_node                               *
186   ******************************************************************************
187   *                                                                             *
188   * Function:      Function that returns a pointer to the next available node.  *
189   *                SMM_NULL_PTR is returned if the avail list is empty.         *
190   *                                                                             *
191   * Input:         smm_get_node();                                              *
192   *                                                                             *
193   * Output:        Pointer (index of) next available node (or SMM_NULL_PTR).    *
194   *                                                                             *
195   * Globals:       smm_error : module SMM.C                                     *
196   *                item      : module SMM.C                                     *
197   *                avail     : module SMM.C                                     *
198   *                                                                             *
199   * Edit History: 02/15/91 - Written by Robin T. Laird.                         *
200   *                                                                             *
201   \*****************************************************************************/
202
203   static nodeptr smm_get_node()
204   {
205     nodeptr p;
206
207     if (avail == SMM_NULL_PTR)
208     {
209       smm_error = SMM_ERR_OVERFLOW;
210       return(SMM_NULL_PTR);
211     }
212     else
213     {
214       smm_error = AOK;
215       p = avail;
216       avail = item[avail].right;
217       return(p);
218     }
219   }
```



```
220
221
222   /*****************************************************************************\
223   *                                  smm_free_node                              *
224   ******************************************************************************
225   *                                                                             *
226   * Function:      Returns the parameter node to the avail list.                *
227   *                An error is generated if the node to be returned is NULL.    *
228   *                                                                             *
229   * Input:         smm_free_node(                                               *
230   *                  nodeptr p; pointer to (index of) node to free.             *
231   *                );                                                           *
232   *                                                                             *
233   * Output:        Nothing.                                                     *
234   *                                                                             *
235   * Globals:       smm_error : module SMM.C                                     *
236   *                item      : module SMM.C                                     *
237   *                avail     : module SMM.C                                     *
238   *                                                                             *
239   * Edit History: 02/15/91 - Written by Robin T. Laird.                         *
240   *                                                                             *
241   \*****************************************************************************/
242
243   static void smm_free_node(p)
244   nodeptr p;
245   {
246     if (p == SMM_NULL_PTR)
247     {
248       smm_error = SMM_ERR_INVALID_PTR;
249     }
250     else
251     {
252       smm_error = AOK;
253       item[p].right = avail;
254       avail = p;
255     }
256   }
257
258
259   /*****************************************************************************\
260   *                                  smm_clear_q                                *
261   ******************************************************************************
262   *                                                                             *
263   * Function:      Initializes the parameter queue and clears its contents.     *
264   *                                                                             *
265   * Input:         smm_clear_q(                                                 *
266   *                  nodeptr *p; pointer to beginning of queue to clear.        *
267   *                );                                                           *
268   *                                                                             *
269   * Output:        Nothing.                                                     *
270   *                                                                             *
271   * Globals:       smm_error : module SMM.C                                     *
272   *                                                                             *
273   * Edit History: 07/09/90 - Written by Robin T. Laird.                         *
274   *                                                                             *
275   \*****************************************************************************/
276
277   static void smm_clear_q(p)
278   nodeptr *p;
279   {
280     smm_error = AOK;                        /* Assume function successful...  */
281     *p = SMM_NULL_PTR;
282   }
283
284
285   /*****************************************************************************\
286   *                                  smm_print_q                                *
287   ******************************************************************************
288   *                                                                             *
289   * Function:      Prints the contents of the parameter queue on the local      *
290   *                output device (serial port). Only the event number and the   *
291   *                associated function ID are currently printed.                *
292   *                                                                             *
```

```
293   * Input:         smm_print_q(                                          *
294   *                   nodeptr *p; pointer to (index of) queue to be printed.  *
295   *                 );                                                    *
296   *                                                                       *
297   * Output:        Nothing.                                              *
298   *                                                                       *
299   * Globals:       item : module SMM.C                                   *
300   *                                                                       *
301   * Edit History: 02/15/91 - Written by Robin T. Laird.                  *
302   *                                                                       *
303   \***********************************************************************/
304
305   static void smm_print_q(p)
306   nodeptr p;
307   {
308     nodeptr q;
309
310     if (p == SMM_NULL_PTR)
311     {
312       #if defined(DEBUG)
313       printf("smm_print_q: NULL list\n\r");
314       #endif
315     }
316     else
317     {
318       q = p;
319       do {
320         #if defined(DEBUG)
321         printf("smm_print_q: l[%02d] : event = %02d fid = %b02d\n\r", q, item[q].
322         #endif
323
324         q = item[q].right;
325       } while (q != p && q != SMM_NULL_PTR);
326     }
327   }
328
329
330   /***********************************************************************\
331   *                              smm_insert_q                             *
332   ************************************************************************
333   *                                                                       *
334   * Function:      Inserts an event into the parameter queue if room.    *
335   *                An error is returned if the queue is already full.    *
336   *                                                                       *
337   * Input:         smm_insert_q(                                         *
338   *                  nodeptr *p;       pointer to queue to insert into.   *
339   *                  mm_message *info; pointer to event to be inserted.   *
340   *                );                                                     *
341   *                                                                       *
342   * Output:        Nothing.                                              *
343   *                                                                       *
344   * Globals:       smm_error : module SMM.C                              *
345   *                item      : module SMM.C                              *
346   *                                                                       *
347   * Edit History: 07/09/90 - Written by Robin T. Laird.                  *
348   *                                                                       *
349   \***********************************************************************/
350
351   static void smm_insert_q(p, info)
352   nodeptr *p;
353   mm_message *info;
354   {
355     nodeptr q, r;
356
357     if ((q = smm_get_node()) != SMM_NULL_PTR)
358     {
359       smm_error = AOK;
360
361       item[q].info = *info;
362
363       if (*p == SMM_NULL_PTR)
364       {
365         item[q].left  = q;
```



```
366         item[q].right = q;
367         *p = q;
368       }
369       else
370       {
371         r = item[*p].left;
372         item[q].left  = r;
373         item[q].right = *p;
374         item[r].right = q;
375         item[*p].left = q;
376       }
377     }
378   }
379
380
381   /***********************************************************************\
382   *                              smm_remove_q                             *
383   ************************************************************************
384   *                                                                       *
385   * Function:      Removes an event from the parameter queue if available.*
386   *                An error is returned if the queue is already empty.    *
387   *                                                                       *
388   * Input:         smm_remove_q(                                         *
389   *                  nodeptr *p;       pointer to queue to remove from.   *
390   *                  mm_message *info; pointer to event that is removed.  *
391   *                );                                                     *
392   *                                                                       *
393   * Output:        Nothing.                                              *
394   *                                                                       *
395   * Globals:       smm_error : module SMM.C                              *
396   *                item      : module SMM.C                              *
397   *                                                                       *
398   * Edit History: 07/09/90 - Written by Robin T. Laird.                  *
399   *                                                                       *
400   \***********************************************************************/
401
402   static void smm_remove_q(p, info)
403   nodeptr *p;
404   mm_message *info;
405   {
406     nodeptr q, r, s;
407
408     if (*p == SMM_NULL_PTR)
409     {
410       smm_error = SMM_ERR_INVALID_PTR;
411     }
412     else
413     {
414       smm_error = AOK;
415
416       *info = item[*p].info;
417
418       q = item[*p].left;
419       r = item[*p].right;
420       s = *p;
421       if (s == q)
422       {
423         *p = SMM_NULL_PTR;
424       }
425       else
426       {
427         *p = item[*p].right;
428         item[q].right = r;
429         item[r].left  = q;
430       }
431       smm_free_node(s);
432     }
433   }
434
435
436   /***********************************************************************\
437   *                              smm_findin_q                             *
438   ************************************************************************
```

```
439   *                                                                       *
440   * Function:     Locates the node in the parameter queue whose event     *
441   *               identifier matches the parameter message's sequence number *
442   *               and whose source address matches the message's destination. *
443   *               The entire queue is searched in a linear fashion since  *
444   *               the queue is not ordered in any particular manner.      *
445   *               If matching event IDs are not found, then SMM_NULL_PTR is *
446   *               returned. Otherwise, the pointer of the node is returned. *
447   *                                                                       *
448   * Input:        smm_findin_q(                                           *
449   *                 nodeptr p;    pointer to (index of) queue to be searched. *
450   *                 mm_message m; message containing search key information. *
451   *               );                                                      *
452   *                                                                       *
453   * Output:       Nothing.                                                *
454   *                                                                       *
455   * Globals:      smm_error : module SMM.C                                *
456   *               item      : module SMM.C                                *
457   *                                                                       *
458   * Edit History: 01/09/90 - Written by Robin T. Laird.                   *
459   *                                                                       *
460   \***********************************************************************/
461
462   static nodeptr smm_findin_q(p, m)
463   nodeptr p;
464   mm_message *m;
465   {
466     nodeptr q;
467
468     if (p == SMM_NULL_PTR)
469     {
470       smm_error = SMM_ERR_INVALID_PTR;
471       return(SMM_NULL_PTR);
472     }
473     else
474     {
475       q = p;
476       do {
477
478         #if defined(DEBUG)
479         printf("smm_findin_q: item[q].event          = %d\n", item[q].event);
480         printf("smm_findin_q: item[q].dest_modbot_id = %d\n", (int)item[q].info.d
481         printf("smm_findin_q: item[q].dest_unit_id   = %d\n", (int)item[q].info.d
482         printf("smm_findin_q: m->sequence_number     = %d\n", (int)m->sequence_nu
483         printf("smm_findin_q: m->src_modbot_id       = %d\n", (int)m->src_modbot_
484         printf("smm_findin_q: m->src_unit_id         = %d\n", (int)m->src_unit_id
485         #endif
486
487         if (item[q].event == m->sequence_number &&
488             item[q].info.dest_modbot_id == m->src_modbot_id &&
489             item[q].info.dest_unit_id   == m->src_unit_id)
490         {
491           smm_error = AOK;
492           return(q);
493         }
494         q = item[q].right;
495       } while (q != p && q != SMM_NULL_PTR);
496       smm_error = SMM_ERR_NO_EVENT_MATCH;
497       return(SMM_NULL_PTR);
498     }
499   }
500
501
502   /***********************************************************************\
503   *                            smm_next_event                             *
504   ************************************************************************
505   *                                                                       *
506   * Function:     Generates a unique sequence number used to identify a   *
507   *               system message. The sequence number must be different   *
508   *               than any other sequence number "currently in use". The  *
509   *               number is used to coorelate response messages to commands. *
510   *                                                                       *
511   * Input:        smm_next_event();                                       *
```



```
512   *                                                                       *
513   * Output:       Returns an int value representing the sequence number.  *
514   *                                                                       *
515   * Globals:      avail : module SMM.C                                    *
516   *                                                                       *
517   * Edit History: 01/23/91 - Written by Robin T. Laird.                   *
518   *                                                                       *
519   \***********************************************************************/
520
521   static int smm_next_event()
522   {
523     return(avail);
524   }
525
526
527   /***********************************************************************\
528   *                                smm_init                               *
529   ************************************************************************
530   *                                                                       *
531   * Function:     Initializes the Local Method Manager software subsystems. *
532   *               This includes initializing the module-specific subsystems *
533   *               via a call to the system function unit_reset().         *
534   *                                                                       *
535   * Input:        smm_init();                                             *
536   *                                                                       *
537   * Output:       Nothing.                                                *
538   *                                                                       *
539   * Globals:      smm_error       : module SMM.C                          *
540   *               avail           : module SMM.C                          *
541   *               queue           : module SMM.C                          *
542   *               smm_num_methods : module SMM.C                          *
543   *                                                                       *
544   * Edit History: 12/20/90 - Written by Robin T. Laird.                   *
545   *                                                                       *
546   \***********************************************************************/
547
548   void smm_init()
549   {
550     smm_error = AOK;                        /* Assume function successful.  */
551
552     /* Reset system method trigger table index to 0 (indicate table empty).  */
553
554     smm_num_methods = 0;
555
556     /* Initialize the free node pool (avail list).                           */
557
558     smm_avail_init(&avail);
559
560     /* Initialize the standard event queue.                                  */
561
562     smm_clear_q(&queue);
563   }
564
565
566   /***********************************************************************\
567   *                              smm_process                              *
568   ************************************************************************
569   *                                                                       *
570   * Function:     Processes the parameter message and updates the local method *
571   *               trigger table. Coordinates message translation and message *
572   *               generation. The input message is processed (translated) and *
573   *               any possible output or response message is generated for *
574   *               transmission to the originating unit.                   *
575   *                                                                       *
576   * Input:        smm_process(                                            *
577   *                 mm_message *m_in;  pointer to decoded message to process. *
578   *                 mm_message *m_out; pointer to possible message to output. *
579   *                 byte *status;      pointer to method execution status. *
580   *               );                                                      *
581   *                                                                       *
582   * Output:       Nothing.                                                *
583   *                                                                       *
584   * Globals:      smm_error : module SMM.C                                *
```

```
585   *                                                                          *
586   * Edit History: 12/20/90 - Written by Robin T. Laird.                      *
587   *                                                                          *
588   \**************************************************************************/
589
590   void smm_process(m_in, m_out, status)
591   mm_message *m_in, *m_out;
592   byte *status;
593   {
594     smm_error = AOK;                          /* Assume function successful.      */
595
596     /* Translate incoming message as response to previous command.             */
597     /* Translation will try and associate sequence number with event number.  */
598
599     smm_translate_message(m_in, m_out, status);
600   }
601
602
603   /**************************************************************************\
604   *                                  smm_fire                                *
605   ****************************************************************************
606   *                                                                          *
607   * Function:     Checks system method trigger table and fires appropriate   *
608   *               functions according to conditions set up in the table.     *
609   *               The conditions are set by internal commands issued by      *
610   *               other functions. The conditions are checked as often as    *
611   *               possible for possible execution of a function.             *
612   *                                                                          *
613   *               Currently, only temporal conditions are implemented.       *
614   *               This allows for periodic execution of functions.           *
615   *                                                                          *
616   * Input:        smm_fire();                                                *
617   *                                                                          *
618   * Output:       Nothing.                                                   *
619   *                                                                          *
620   * Globals:      smm_error        : module SMM.C                            *
621   *               smm_trigger      : module SMM.C                            *
622   *               smm_num_methods  : module SMM.C                            *
623   *                                                                          *
624   * Edit History: 03/28/91 - Written by Robin T. Laird.                      *
625   *                                                                          *
626   \**************************************************************************/
627
628   #define SMM_SEND_ATTEMPTS       LCI_WAIT_FOREVER
629
630   void smm_fire()
631   {
632     int  i;
633     byte event;
634
635     smm_error = AOK;                          /* Assume function successful...    */
636
637     /* Check number of methods in trigger table. If non-zero, continue.       */
638
639     if (smm_num_methods)
640     {
641       /* Check each method for activation.                                     */
642       /* Compare current time to trigger time. If greater, then fire method.  */
643
644       for (i = 0; i < smm_num_methods; i++)
645       {
646         if (rtc_time() > smm_trigger[i].fireat)
647         {
648           /* Process method as usual.                                          */
649           /* Update next fire time.                                            */
650
651           smm_trigger[i].fireat = smm_trigger[i].period+rtc_time();
652         }
653       }
654     }
655   }
656
657
```



```
658   /**************************************************************************\
659   *                              smm_generate_message                        *
660   ****************************************************************************
661   *                                                                          *
662   * Function:     Generates a message as the result of a library function call.*
663   *               The function call causes a "method activation record" to be *
664   *               placed on the system method queue (if space is available).  *
665   *               A message is assembled and sent, and an event identifier    *
666   *               in the form of a newly generated sequence number is returned*
667   *               to the routine that invoked the library function.           *
668   *                                                                          *
669   *               If a destination unit name is supplied (NOT NULL), then the *
670   *               address of the named unit is looked up in the phone book and*
671   *               used in the generated message.                              *
672   *                                                                          *
673   * Input:        smm_generate_message(                                      *
674   *                 char *dest_unit_name; pointer to name of unit destination.*
675   *                 mm_message *m_in;     pointer to (partial) message input. *
676   *                 int *event;           pointer to returned event identifier.*
677   *               );                                                         *
678   *                                                                          *
679   * Output:       Nothing.                                                   *
680   *                                                                          *
681   * Globals:      smm_error : module SMM.C                                   *
682   *               queue     : module SMM.C                                   *
683   *               mm_stdout : module MM.C                                    *
684   *               lci_error : module LCI.C                                   *
685   *                                                                          *
686   * Edit History: 12/20/90 - Written by Robin T. Laird.                      *
687   *                                                                          *
688   \**************************************************************************/
689
690   #define SMM_SEND_ATTEMPTS       LCI_WAIT_FOREVER
691
692   void smm_generate_message(dest_unit_name, m_in, event)
693   char *dest_unit_name;
694   mm_message *m_in;
695   int *event;
696   {
697     byte modbot_addr, unit_addr;
698     lci_message l_out;
699
700     smm_error = AOK;                          /* Assume function successful.      */
701
702     /* Make sure we can add another event to the method manager queue.         */
703
704     if (smm_full_q(queue))
705     {
706       smm_error = SMM_ERR_FULL_QUEUE;
707       *event    = MM_NULL_EVENT;
708       return;
709     }
710
711     /* If destination unit name is NULL, then address is suppiled in m_in.    */
712     /* Else look up destination unit name in the phonebook and get address.   */
713     /* If we can't find it, then return NULL event indicating failure.        */
714
715     if (dest_unit_name != NULLPTR)
716     {
717       if (!pb_lookup(dest_unit_name, &modbot_addr, &unit_addr))
718       {
719         smm_error = SMM_ERR_MSG_GENERATION;
720         *event    = MM_NULL_EVENT;
721         return;
722       }
723       else
724       {
725         m_in->dest_modbot_id = modbot_addr;
726         m_in->dest_unit_id   = unit_addr;
727       }
728     }
729
730     /* Generate the rest of the outgoing message.                              */
```

```
731     /* Assumes the following fields have been filled accordingly:              */
732     /*   dest modbot id            :       above                               */
733     /*   dest unit id              :       above                               */
734     /*   message_length            :       supplied (generated by mm_encode)   */
735     /*   src_modbot_id             :       supplied (determined by GCI)        */
736     /*   src_unit_id               :       supplied (determined by GCI)        */
737     /*   sequence_number           :       generated below                     */
738     /*   trans_disposition         :       supplied (m_in)                     */
739     /*   trans_category            :       supplied (m_in)                     */
740     /*   function_id               :       supplied (m_in)                     */
741     /*   parameter_length          :       global (mm_stdout)                  */
742     /*   parameter                 :       global (mm_stdout)                  */
743     /* Note that we have to adjust for bit output by testing bit index.        */
744
745     *event = smm_next_event();
746
747     if (mm_stdout.bitindex) mm_stdout.index++;
748     m_in->sequence_number   = (byte)*event;
749     m_in->parameter_length = mm_stdout.index;
750     memcpy(m_in->parameter, mm_stdout.buffer, m_in->parameter_length);
751     mm_stdout.index     = 0;
752     mm_stdout.bitindex = 0;
753
754     /* Encode the above information for transmission via the LCI.              */
755     /* And send it...                                                          */
756
757     mm_encode_message(m_in, l_out);
758     lci_send_message(l_out, SMM_SEND_ATTEMPTS);
759
760     /* Change disposition of message to indicate awaiting response.           */
761
762     m_in->trans_disposition = MM_AWAITING_EVENT;
763
764     /* If msg sent OK, then add method activation record to queue.            */
765
766     if (lci_error == AOK) smm_insert_q(&queue, m_in);
767   }
768
769
770   /*************************************************************************\
771   *                          smm_translate_message                          *
772   *************************************************************************
773   *                                                                         *
774   * Function:     Translates (copies) the parameter input message (m_in) into *
775   *               the system method queue at the position indicated by the  *
776   *               event parameter. The method is then available for servicing *
777   *               by the application process (routine).                     *
778   *                                                                         *
779   * Input:        smm_translate_message(                                    *
780   *                 mm_message *m_in;  pointer to message to translate.     *
781   *                 mm_message *m_out; pointer to any response message.     *
782   *                 byte *status;      pointer to translation status.       *
783   *               );                                                        *
784   *                                                                         *
785   * Output:       Nothing.                                                  *
786   *                                                                         *
787   * Globals:      smm_error : module SMM.C                                  *
788   *               queue     : module SMM.C                                  *
789   *               item      : module SMM.C                                  *
790   *                                                                         *
791   * Edit History: 02/15/91 - Written by Robin T. Laird.                     *
792   *                                                                         *
793   \*************************************************************************/
794
795   void smm_translate_message(m_in, m_out, status)
796   mm_message *m_in, *m_out;
797   byte *status;
798   {
799     nodeptr n;
800
801     /* Associate sequence number of incoming msg with msgs in event queue.   */
802     /* Search queue starting at front for matching sequence number.          */
803
```



```
804     if ((n = smm_findin_q(queue, m_in)) == SMM_NULL_PTR)
805     {
806       smm_error = SMM_ERR_NO_EVENT_MATCH;
807       *status   = MM_NO_RESPONSE;
808     }
809     else
810     {
811       /* Found matching sequence number, so translate input message.         */
812       /* Translation will cause method queue to be updated.                  */
813       /* This is where the transaction disposition changes.                  */
814
815       /* Copy message info into method queue at position indicated by event. */
816       /* The item in the queue that corresponds to the event number - that is */
817       /* based on the input message sequence number - will be changed.       */
818       /* In particular, the transaction disposition and transaction category. */
819
820       item[n].info.message_length    = m_in->message_length;
821       item[n].info.trans_disposition = m_in->trans_disposition;
822       item[n].info.trans_category    = m_in->trans_category;
823       item[n].info.parameter_length  = m_in->parameter_length;
824       memcpy(item[n].info.parameter,m_in->parameter,(int)m_in->parameter_length);
825
826       smm_error = AOK;
827       *status   = MM_NO_RESPONSE;
828     }
829   }
```

```
/*****************************************************************************\
*                                  SMMLIB.C                                   *
*****************************************************************************
*                                                                            *
* CPCI:          IED90-MRA-COM-MMS-SMMLIB-C-R0C0                             *
*                                                                            *
* Description:   System Method Manager (SMM) library functions.              *
*                Implements the inherited ("built-in") system methods.       *
*                                                                            *
*                Module SMMLIB.C exports the following functions:            *
*                                                                            *
*                OBJFCT (STATUS_REQUEST):                                    *
*                obj_class();                                                *
*                obj_superclass();                                           *
*                                                                            *
*                UNIT (STATUS_REQUEST):                                      *
*                unit_name();                                                *
*                                                                            *
*                UNIT (CONTROL, CONTROL_WITH_ACK):                           *
*                unit_reset();                                               *
*                                                                            *
* Notes:         None.                                                       *
*                                                                            *
* Edit History: 01/22/91 - Written by Robin T. Laird.                        *
*                                                                            *
\*****************************************************************************/

#include <sysdefs.h>                    /* System constants and types.      */
#include "mm.h"                         /* Method manager literals.         */
#include "smm.h"                        /* System method manager.           */

/* Object class library functions...                                         */

int obj_class(modbot, unit)
byte modbot, unit;
{
  int         event;
  mm_message m;

  m.dest_modbot_id     = modbot;
  m.dest_unit_id       = unit;
  m.trans_disposition = MM_INITIATING;
  m.trans_category     = MM_STATUS_REQUEST;
  m.function_id        = OBJ_FN_CLASS;
  smm_generate_message(NULLPTR, &m, &event);
  return(event);
}

int obj_superclass(modbot, unit)
byte modbot, unit;
{
  int         event;
  mm_message m;

  m.dest_modbot_id     = modbot;
  m.dest_unit_id       = unit;
  m.trans_disposition = MM_INITIATING;
  m.trans_category     = MM_STATUS_REQUEST;
  m.function_id        = OBJ_FN_SUPERCLASS;
  smm_generate_message(NULLPTR, &m, &event);
  return(event);
}

/* Unit class library functions...                                           */

int unit_name(modbot, unit)
byte modbot, unit;
{
  int         event;
  mm_message m;

  m.dest_modbot_id     = modbot;
  m.dest_unit_id       = unit;
```



```
  m.trans_disposition = MM_INITIATING;
  m.trans_category     = MM_STATUS_REQUEST;
  m.function_id        = UNIT_FN_NAME;
  smm_generate_message(NULLPTR, &m, &event);
  return(event);
}

int unit_reset(modbot, unit)
byte modbot, unit;
{
  int         event;
  mm_message m;

  m.dest_modbot_id     = modbot;
  m.dest_unit_id       = unit;
  m.trans_disposition = MM_INITIATING;
  m.trans_category     = MM_CONTROL_WITH_ACK;
  m.function_id        = UNIT_FN_RESET;
  smm_generate_message(NULLPTR, &m, &event);
  return(event);
}
```

```
#*****************************************************************************#
#                                  MAKEFILE                                   #
#*****************************************************************************#
#                                                                             #
# CPCI:          IED90-MRA-ICN-AC-MAKEFILE-TXT-R0C0                           #
#                                                                             #
# Description:   Makefile for the Modular Robotic Architecture (MRA).         #
#                Makes the ICN applications controller subsystem.             #
#                                                                             #
#                Targets are available for the following systems/subsystems:  #
#                                                                             #
#                iac         - ICN Applications Controller                    #
#                iac.hex     - ICN Applications Controller Program (80152)    #
#                icnmor.hex  - ICN Network Monitor Program (80152)            #
#                print       - Print ICN AC source files                      #
#                                                                             #
# Notes:         1) The dependency and production rules are included here.     #
#                2) See also \mra\makefile.                                   #
#                                                                             #
# Edit History: 03/25/91 - Written by Robin T. Laird.                         #
#                                                                             #
#*****************************************************************************#


################################### RULES ###################################

.SUFFIXES : .hex .exe .obj .c .a51

# Control settings for Franklin 8031 development

CC          =c51
AS          =a51
LINK        =l51
OTOH        =ohs51
CFLAGS      =cd la db sb
ASFLAGS     =
LFLAGS      =
OFLAGS      =
STARTUP     =\c51\crom.obj
CODESEG     =00000h
XDATASEG    =00000h

# Control settings for Microsoft MS-DOS development

MSC         =cl
MSAS        =masm
MSLINK      =link
MSCFLAGS    =/AS /c /O1 /Zi /Od
MSASFLAGS   =
MSLNKFLGS   =/co
LOADLIBES   =

.c.obj :
        $(CC) $< $(CFLAGS)

.a51.obj :
        $(AS) $< $(ASFLAGS)

.obj.exe :
        $(LINK) $(STARTUP),$< TO $@ code ($(CODESEG)) xdata ($(XDATASEG)) ixref

.exe.hex :
        $(OTOH) $< $(OFLAGS)


################################ DEFINITIONS ################################

# Project, system, and application level definitions

PROJ            = mra
APPSYS          = app
```



```
COMSYS          = com
ICNSYS          = icn
MPUSYS          = mpu

APPSRC          = \$(PROJ)\$(APPSYS)\src
COMSRC          = \$(PROJ)\$(COMSYS)\src
ICNSRC          = \$(PROJ)\$(ICNSYS)\src
MPUSRC          = \$(PROJ)\$(MPUSYS)\src

COMBIN152       = \$(PROJ)\$(COMSYS)\bin\80152

ICNBIN152       = \$(PROJ)\$(ICNSYS)\bin\80152
ICNBIN152_MON   = \$(PROJ)\$(ICNSYS)\bin\80152\mon

# Common subsystem level source directories

DEVSRC          = $(COMSRC)\dev
HDRSRC          = $(COMSRC)\hdr
LCSSRC          = $(COMSRC)\lcs
MMSSRC          = $(COMSRC)\mms

# ICN subsystem level source directories

GCSSRC          = $(ICNSRC)\gcs
IACSRC          = $(ICNSRC)\ac

# MPU subsystem level source directories

LDSSRC          = $(MPUSRC)\lds
MACSRC          = $(MPUSRC)\ac

# Common subsystem global include and compilation units

SYSDEFS         = $(HDRSRC)\sysdefs.h

# ICN subsystem compilation units

IAC             = $(ICNBIN152)\main.obj         $(ICNBIN152)\mra.obj    \
                  $(ICNBIN152)\gci.obj          $(ICNBIN152)\gcd.obj    \
                  $(COMBIN152)\lci.obj          $(COMBIN152)\lcd.obj

ICNMON          = $(ICNBIN152)\main.obj         $(ICNBIN152)\mra.obj    \
                  $(ICNBIN152)\gci.obj          $(ICNBIN152_MON)\gcd.obj\
                  $(COMBIN152)\lci.obj          $(COMBIN152)\lcd.obj


################################## TARGETS ##################################

iac                         : $(ICNBIN152)\iac.hex $(ICNBIN152_MON)\icnmon.hex

$(ICNBIN152)\iac.hex        : $(ICNBIN152)\iac
                              $(OTOH) $< hex($@)

$(ICNBIN152)\iac            : $(IAC)
                              $(LINK) @$(IACSRC)\iac.res

$(ICNBIN152_MON)\icnmon.hex : $(ICNBIN152_MON)\icnmon
                              $(OTOH) $< hex($@)

$(ICNBIN152_MON)\icnmon     : $(ICNMON)
                              $(LINK) @$(IACSRC)\icnmon.res

print                       : $(IAC)
                              %-a2ps -nf main.c | post
                              %-a2ps -nf mra.h | post
                              %-a2ps -nf mra.c | post
                              touch print


############################ ICN IAC DEPENDENCIES ############################
```

```
147   # ICN IAC main program dependencies
148
149   $(ICNBIN152)\main.obj        : $(SYSDEFS)             \
150                                  $(IACSRC)\mra.h        \
151                                  $(GCSSRC)\gci.h        \
152                                  $(GCSSRC)\gcd.h        \
153                                  $(ICSSRC)\lci.h        \
154                                  $(LCSSRC)\lcd.h        \
155                                  $(MMSSRC)\mm.h         \
156                                  $(MMSSRC)\lmm.h        \
157                                  $(MMSSRC)\smm.h        \
158                                  $(LDSSRC)\ldi.h        \
159                                  $(IACSRC)\main.c
160     $(CC) $(IACSRC)\$*.c $(CFLAGS) df(180152) pr($(IACSRC)\$*.152) oj($(ICNBIN152
161
162
163   # ICN IAC mra system dependencies
164
165   $(ICNBIN152)\mra.obj         : $(SYSDEFS)             \
166                                  $(IACSRC)\mra.h        \
167                                  $(GCSSRC)\gci.h        \
168                                  $(GCSSRC)\gcd.h        \
169                                  $(LCSSRC)\lci.h        \
170                                  $(LCSSRC)\lcd.h        \
171                                  $(MMSSRC)\mm.h         \
172                                  $(MMSSRC)\lmm.h        \
173                                  $(MMSSRC)\smm.h        \
174                                  $(LDSSRC)\ldi.h        \
175                                  $(IACSRC)\mra.c
176     $(CC) $(IACSRC)\$*.c $(CFLAGS) df(180152) pr($(IACSRC)\$*.152) oj($(ICNBIN152
```

```
1     \c51\crom.obj,                &
2     \mra\icn\bin\80152\main.obj,  &
3     \mra\icn\bin\80152\mra.obj,   &
4     \mra\lib\mra_1521.lib         &
5     to \mra\icn\bin\80152\iac     &
6     pr(\mra\icn\src\ac\iac.m51) co(00000h) xd(00000h) ix
```

```
1     \c51\crom.obj,                  &
2     \mra\icn\bin\80152\main.obj,    &
3     \mra\icn\bin\80152\mra.obj,     &
4     \mra\icn\bin\80152\gc1.obj,     &
5     \mra\icn\bin\80152\mon\gcd.obj, &
6     \mra\lib\mra_1571.lib           &
7     to \mra\icn\bin\80152\mon\icnmon&
8     pr(\mra\icn\src\ac\icnmon.m51) co(00000h) xd(00000h) ix
```

```
1   /*****************************************************************************\
2   *                                  MAIN.C                                     *
3   *****************************************************************************
4   *                                                                            *
5   * CPCI:          IED90-MRA-ICN-AC-MAIN-C-R0C0                                *
6   *                                                                            *
7   * Description:   ICN main program.                                           *
8   *                Implements the MRA ICN (user) application program.          *
9   *                This is the main program for the ICN system.                *
10  *                It simply calls the standard MRA function mra_init() which  *
11  *                is responsible for initializing and coordinating the MRA    *
12  *                subsystems for the ICN application.                         *
13  *                                                                            *
14  * Notes:         1) Only the LCS and GCS subsystems are currently required.  *
15  *                2) Subsystems are selected by setting the associated USE_ss *
16  *                   variable to YES (1). This can be done from either the    *
17  *                   compilation command line or by modifying the MRA.H file. *
18  *                                                                            *
19  * Edit History: 10/10/90 - Written by Robin T. Laird.                        *
20  *                                                                            *
21  \*****************************************************************************/
22
23  #include <sysdefs.h>                    /* MRA standard declarations.       */
24  #include "mra.h"                        /* MRA public literals/functions.   */
25
26  void main()
27  {
28    /* Initialize the MRA subsystems and call mra_main().                      */
29    /* Control never returns...                                                */
30
31    mra_init();
32  }
```

```
1   /*************************************************************************\
2   *                                MRA.H                                    *
3   *************************************************************************
4   *                                                                       *
5   * CPCI:          IED90-MRA-ICN-AC-MRA-H-R0C0                            *
6   *                                                                       *
7   * Description:   System configuration and default controller definitions.*
8   *                Contains external declarations for the system initialization *
9   *                function and default application controller, mra_main(). *
10  *                                                                       *
11  *                The user/developer should select/de-select those MRA   *
12  *                subsystems that are required or being used. Only those *
13  *                subsystems that are selected are initialized by mra_init(). *
14  *                                                                       *
15  *                Selecting USE_MRA will cause the default system application *
16  *                controller, mra_main(), to be used. The init routine calls *
17  *                the default controller after all selected subsystems have *
18  *                been initialized. If selected, mra_main() takes control and *
19  *                does not return.                                       *
20  *                                                                       *
21  *                Module MRA exports the following variables/functions:  *
22  *                                                                       *
23  *                int mra_error;                                         *
24  *                                                                       *
25  *                mra_init();                                            *
26  *                mra_main();                                            *
27  *                                                                       *
28  * Notes:         1) This file should be included only by the main() program. *
29  *                                                                       *
30  * Edit History: 07/07/90 - Written by Robin T. Laird.                   *
31  *                                                                       *
32  \*************************************************************************/
33
34  #ifndef MRA_MODULE_CODE
35  #define MRA_MODULE_CODE         12000
36
37  /* Public Data Structures:                                              */
38
39  #define ERR_MRA_NOT_INIT        1+MRA_MODULE_CODE
40
41  #define USE_GCS                 YES     /* Global Communications Subsystem. */
42  #define USE_LCS                 YES     /* Local  Communications Subsystem. */
43  #define USE_MMS                 NO      /* Method Management Subsystem.     */
44  #define USE_LDS                 NO      /* Logical Device Subsystem.        */
45  #define USE_MRA                 YES     /* Modular Robotic Architecture AC. */
46
47  #if USE_GCS
48  #include <gci.h>                        /* Global Comms Interface.          */
49  #include <gcd.h>                        /* Global Comms Device Driver.      */
50  #endif
51
52  #if USE_LCS
53  #include <lci.h>                        /* Local Comms Interface.           */
54  #include <lcd.h>                        /* Local Comms Device Driver.       */
55  #endif
56
57  #if USE_MMS
58  #include <mm.h>                         /* Method Manager.                  */
59  #include <pb.h>                         /* Phone Book.                      */
60  #include <lmm.h>                        /* Local Method Manager.            */
61  #include <smm.h>                        /* System Method Manager.           */
62  #endif
63
64  #if USE_LDS
65  #include <ldi.h>                        /* Logical Device Interface.        */
66  #endif
67
68  /* External module global error variable.                               */
69
70  extern int mra_error;
71
72  /* Public Functions:                                                    */
73
```



```
74  void mra_init(void);
75  void mra_main(void);
76
77  #endif
```

```
1   /*****************************************************************************\
2   *                                  MRA.C                                      *
3   ******************************************************************************
4   *                                                                             *
5   * CPCI:          IED90-MRA-ICN-AC-MRA-C-R0C0                                  *
6   *                                                                             *
7   * Description:   MRA system initialization and default controller functions.  *
8   *                Implements the MRA system initialization and default system  *
9   *                application controller (AC) functions which represent the    *
10  *                highest level interface to the Modular Robotic Architecture  *
11  *                software systems. The mra_init() function must be called to  *
12  *                correctly initialize the various software subsystems. The    *
13  *                function mra_main() is the default AC and replaces the user  *
14  *                application program (mra_main() never returns to the calling *
15  *                function).                                                   *
16  *                                                                             *
17  *                Module MRA exports the following variables/functions:        *
18  *                                                                             *
19  *                int mra_error;                                               *
20  *                                                                             *
21  *                mra_init();                                                  *
22  *                mra_main();                                                  *
23  *                                                                             *
24  * Notes:         1) The MRA functions are implementation independent.         *
25  *                2) Module MRA represents the Default Applications Controller.*
26  *                                                                             *
27  * Edit History: 07/28/90 - Written by Robin T. Laird.                         *
28  *                                                                             *
29  \*****************************************************************************/
30
31  #include <sysdefs.h>                    /* System constants and types.      */
32  #include "mra.h"                        /* MRA public literals/functions.   */
33
34  /* Public Variables:                                                        */
35
36  /* Global module error variable, mra_error.                                */
37  /* mra_error contains code of last error occurence.                         */
38  /* Should be set to AOK after each successful function call.                */
39  /* Variable can be examined by other software after each function call.     */
40
41  XDATA int mra_error = ERR_MRA_NOT_INIT;
42
43
44  /*****************************************************************************\
45  *                                 mra_init                                    *
46  ******************************************************************************
47  *                                                                             *
48  * Function:      Initializes the MRA software subsystems.                     *
49  *                The subsystems are selected in the file MRA.H and only those *
50  *                subsystems selected will be initialized. If the default      *
51  *                application controller mra_main() is selected then a call is *
52  *                made to that function and control never returns. The default *
53  *                AC can be called separately by a call to mra_main() after    *
54  *                mra_init() returns (the same result is acheived).            *
55  *                                                                             *
56  * Input:         mra_init();                                                  *
57  *                                                                             *
58  * Output:        Nothing.                                                     *
59  *                                                                             *
60  * Globals:       mra_error : module MRA.C                                     *
61  *                gci_error : module GCI.C                                     *
62  *                lci_error : module LCI.C                                     *
63  *                mm_error  : module MM.C                                      *
64  *                ldi_error : module LDI.C                                     *
65  *                                                                             *
66  * Edit History: 10/01/90 - Written by Robin T. Laird.                         *
67  *                                                                             *
68  \*****************************************************************************/
69
70  void mra_init()
71  {
72    /* Initialize selected subsystems, return upon detected failure.          */
73
```



```
74    #if USE_GCS
75    gci_init();
76    if (gci_error != AOK) return;
77    #endif
78
79    #if USE_LCS
80    lci_init();
81    if (lci_error != AOK) return;
82    #endif
83
84    #if USE_MMS
85    mm_init();
86    if (mm_error != AOK) return;
87    #endif
88
89    #if USE_LDS
90    ldi_init();
91    if (ldi_error != AOK) return;
92    #endif
93
94    /* Function successful...                                                 */
95    /* Set error variable to OK and return only if USE_MRA not selected.      */
96
97    mra_error = AOK;
98
99    /* Call MRA main program and never return...                              */
100
101   #if USE_MRA
102   mra_main();
103   #endif
104 }
105
106
107 /*****************************************************************************\
108 *                                 mra_main                                    *
109 ******************************************************************************
110 *                                                                             *
111 * Function:      Default Application Controller (AC) for the ICN system.      *
112 *                The main() C program calls mra_init() and which then calls   *
113 *                mra_main(). The default controller coordinates operation of  *
114 *                the MRA subsystems to pass information from the LAN to the   *
115 *                module processor (MPU) and vice versa.                       *
116 *                                                                             *
117 * Input:         mra_main();                                                  *
118 *                                                                             *
119 * Output:        Nothing.                                                     *
120 *                                                                             *
121 * Globals:       None.                                                        *
122 *                                                                             *
123 * Edit History: 10/31/90 - Written by Robin T. Laird.                         *
124 *                                                                             *
125 \*****************************************************************************/
126
127 #define MRA_LOCAL_SEND_ATTEMPTS         LCI_WAIT_FOREVER
128 #define MRA_GLOBAL_SEND_ATTEMPTS        8
129
130 void mra_main()
131 {
132   #if USE_MRA
133
134   gci_message g;
135   lci_message l;
136   lcd_state   s;
137   int         gci_rcv_error, lci_rcv_error;
138
139   /* Do forever...                                                          */
140
141   /* Get message from network (ICN LAN).                                    */
142   /* Function will return immediately if nothing available.                 */
143   /* Save GCI global error state.                                           */
144
145   /* Get message from module processor (MPU).                               */
146   /* Function will return immediately if nothing available.                 */
```

```
147     /* Save LCI global error state.                                      */
148
149     /* If no error on receipt of network msg then pass it on to MPU.     */
150     /* If no error on receipt of MPU msg then pass it on to ICN LAN.     */
151
152     while (1)
153     {
154       gci_receive_message(g, GCI_DONT_WAIT);
155       gci_rcv_error = gci_error;
156
157       lci_receive_message(l, LCI_DONT_WAIT);
158       lci_rcv_error = lci_error;
159
160       if (gci_rcv_error == AOK)
161       {
162         lci_send_message((lci_message)g, MRA_LOCAL_SEND_ATTEMPTS);
163       }
164
165       if (lci_rcv_error == AOK)
166       {
167         gci_send_message((gci_message)l, MRA_GLOBAL_SEND_ATTEMPTS);
168       }
169     }
170
171     #endif
172   }
```

```
1   #*****************************************************************************#
2   #                                  MAKEFILE                                   #
3   #*****************************************************************************#
4   #                                                                             #
5   # CPCI:          IED90-MRA-ICN-GCS-MAKEFILE-TXT-R0C0                          #
6   #                                                                             #
7   # Description:   Makefile for the Modular Robotic Architecture (MRA).        #
8   #                Makes the ICN global communications subsystem.               #
9   #                                                                             #
10  #                Targets are available for the following systems/subsystems:  #
11  #                                                                             #
12  #                gcs   - ICN Global Communications Subsystem                  #
13  #                lib   - Add modules to COM library                           #
14  #                print - Print ICN GCS source files                           #
15  #                                                                             #
16  # Notes:         1) The dependency and production rules are included here.    #
17  #                2) See also \mra\makefile.                                   #
18  #                                                                             #
19  # Edit History: 03/22/91 - Written by Robin T. Laird.                         #
20  #                                                                             #
21  #*****************************************************************************#
22
23
24
25  ##################################### RULES #####################################
26
27  .SUFFIXES : .hex .exe .obj .c .a51
28
29  # Control settings for Franklin 8031 development
30
31  CC          =c51
32  AS          =a51
33  LINK        =l51
34  OTOH        =ohs51
35  CFLAGS      =cd la db sb
36  ASFLAGS     =
37  LFLAGS      =
38  OFLAGS      =
39  STARTUP     =\c51\crom.obj
40  CODESEG     =00000h
41  XDATASEG    =00000h
42
43  # Control settings for Microsoft MS-DOS development
44
45  MSC         =cl
46  MSAS        =masm
47  MSLINK      =link
48  MSCFLAGS    =/AS /c /Oi /Zi /Od
49  MSASFLAGS   =
50  MSLNKFLGS   =/co
51  LOADLIBES   =
52
53  .c.obj :
54          $(CC) $< $(CFLAGS)
55
56  .a51.obj :
57          $(AS) $< $(ASFLAGS)
58
59  .obj.exe :
60          $(LINK) $(STARTUP),$< TO $@ code ($(CODESEG)) xdata ($(XDATASEG)) ixref
61
62  .exe.hex :
63          $(OTOH) $< $(OFLAGS)
64
65
66
67  ################################### DEFINITIONS ###################################
68
69  # Project, system, and application level definitions
70
71  PROJ            = mra
72  APPSYS          = app
73  COMSYS          = com
```



```
74  ICNSYS          = icn
75  MPUSYS          = mpu
76
77  COMSRC          = \$(PROJ)\$(COMSYS)\src
78
79  ICNLIB          = \$(PROJ)\lib
80  ICNSRC          = \$(PROJ)\$(ICNSYS)\src
81  ICNBIN152       = \$(PROJ)\$(ICNSYS)\bin\80152
82  ICNBIN152_MON   = \$(PROJ)\$(ICNSYS)\bin\80152\mon
83
84  # Common subsystem level source directories
85
86  HDRSRC          = $(COMSRC)\hdr
87
88  # ICN subsystem level source directories
89
90  GCSSRC          = $(ICNSRC)\gcs
91
92  # Common subsystem global include and compilation units
93
94  SYSDEFS         = $(HDRSRC)\sysdefs.h
95
96  # ICN subsystem compilation units
97
98  GCS             = $(ICNBIN152)\gcd.obj  $(ICNBIN152)\gci.obj            \
99                    $(ICNBIN152_MON)\gcd.obj
100
101
102
103 ################################### TARGETS ###################################
104
105 gcs             : $(GCS)
106
107 lib             : $(GCS)
108                   -lib51 delete $(ICNLIB)\mra_1521.lib (gcd, gci)
109                   -lib51 add $(ICNBIN152)\gcd.obj to $(ICNLIB)\mra_1521.lib
110                   -lib51 add $(ICNBIN152)\gci.obj to $(ICNLIB)\mra_1521.lib
111                   touch lib
112
113 print           : $(GCS)
114                   %-a2ps -nf bmf.h | post
115                   %-a2ps -nf bmf.c | post
116                   %-a2ps -nf gcd.h | post
117                   %-a2ps -nf gcd.c | post
118                   %-a2ps -nf gci.h | post
119                   %-a2ps -nf gci.c | post
120                   %-a2ps -nf gsc.h | post
121                   %-a2ps -nf gsc.c | post
122                   touch print
123
124
125
126 ############################## ICN GCS DEPENDENCIES ##############################
127
128 # ICN GCD module dependencies for generic ICN and ICN monitor
129
130 GCD = $(SYSDEFS)        $(GCSSRC)\bmf.h         $(GCSSRC)\bmf.c         \
131       $(GCSSRC)\gsc.h   $(GCSSRC)\gsc.c         $(GCSSRC)\gcd.h         \
132       $(GCSSRC)\gcd.c
133
134 $(ICNBIN152)\gcd.obj     : $(GCD)
135   $(CC) $(GCSSRC)\$*.c $(CFLAGS) df(I80152) pr($(GCSSRC)\$*.152) oj($(ICNBIN152
136
137 $(ICNBIN152_MON)\gcd.obj : $(GCD)
138   $(CC) $(GCSSRC)\$*.c $(CFLAGS) df(ICNMON) pr($(GCSSRC)\$*.mon) oj($(ICNBIN152
139
140
141 # ICN GCI module dependencies
142
143 $(ICNBIN152)\gci.obj     : $(SYSDEFS)                                   \
144                            $(GCSSRC)\gcd.h                              \
145                            $(GCSSRC)\gci.h                              \
146                            $(GCSSRC)\gci.c
```

```
147     $(CC) $(GCSSRC)\$*.c $(CFLAGS) df(180152) pr($(GCSSRC)\$*.152) oj($(ICNBIN152
```

```
1   /*****************************************************************************\
2   *                                  BMF.H                                      *
3   *****************************************************************************
4   *                                                                            *
5   * CPCI:          IED90-MRA-ICN-GCS-BMF-H-R0C1                                *
6   *                                                                            *
7   * Description:   Frame Buffer Management variables and definitions.          *
8   *                Contains constant declarations and type definitions for the *
9   *                circular frame buffer management functions. Used by the     *
10  *                GCS communications device handler module.                   *
11  *                                                                            *
12  *                Module BMF exports the following types/functions:           *
13  *                                                                            *
14  *                typedef buffer;                                             *
15  *                                                                            *
16  *                buf_full();                                                 *
17  *                buf_empty();                                                *
18  *                buf_clear();                                                *
19  *                buf_insert();                                               *
20  *                buf_remove();                                               *
21  *                                                                            *
22  * Notes:         1) This file is included by GCD.C.                          *
23  *                2) Module BMF requires type definitions from GCD.H.         *
24  *                                                                            *
25  * Edit History: 06/28/90 - Written by Robin T. Laird.                        *
26  *                                                                            *
27  \*****************************************************************************/
28
29  /* Private Data Structures:                                                  */
30
31  #ifndef BMF_MODULE_CODE
32  #define BMF_MODULE_CODE          1100
33
34  #define ERR_FULL_BUFFER          1+BMF_MODULE_CODE
35  #define ERR_EMPTY_BUFFER         2+BMF_MODULE_CODE
36
37  #define ERR_SRC_EQUAL_DEST       3+BMF_MODULE_CODE
38
39  /* MAX_BUFFER_SIZE defines the number of receive/transmit frames/buffer.    */
40
41  #define MAX_BUFFER_SIZE          48
42
43  /* Circular buffer (queue) to hold incoming/outgoing data frames.           */
44  /* Must be initialized using buf_clear().                                   */
45
46  typedef struct { gcd_frame item[MAX_BUFFER_SIZE];
47                   byte      front;
48                   byte      rear;
49                   byte      empty;
50                   byte      full;
51                 } buffer;
52
53  /* Private Functions:                                                       */
54
55  static int buf_full(buffer *b);
56  static int buf_empty(buffer *b);
57  static void buf_clear(buffer *b);
58  static void buf_insert(buffer *b, gcd_frame f);
59  static void buf_remove(buffer *b, gcd_frame f);
60
61  #endif
```

```
/*****************************************************************************\
*                                   BMF.C                                     *
*****************************************************************************
*                                                                           *
* CPCI:          IED90-MRA-ICN-GCS-BMF-C-ROC2                               *
*                                                                           *
* Description:   Frame Buffer Management functions.                         *
*                Contains functions for initializing and managing the       *
*                circular frame buffers for the GCS global communications   *
*                device handler.                                            *
*                                                                           *
*                Module BMF exports the following functions:                *
*                                                                           *
*                buf_front() macro;                                         *
*                buf_rear()  macro;                                         *
*                buf_inc()   macro;                                         *
*                buf_full();                                                *
*                buf_empty();                                               *
*                buf_clear();                                               *
*                buf_insert();                                              *
*                buf_remove();                                              *
*                                                                           *
* Notes:         1) This module is NOT a stand-alone compilation unit.      *
*                   It is included by the module GCD.C and is compiled there.*
*                   It is assumed that the file GCD.H is included before it.*
*                   Note that all of the functions herein are static.       *
*                                                                           *
* Edit History: 06/29/90 - Written by Richard P. Smurlo and Robin T. Laird. *
*                                                                           *
\*****************************************************************************/

/* Private Variables:                                                       */

/* Declarations for the module circular receive and transmit buffers.       */
/* These are global to the GSC.C module and to the GCD.C module.            */

static XDATA buffer xmt_buffer;
static XDATA buffer rcv_buffer;


/*****************************************************************************\
*                                  buf_front                                  *
*****************************************************************************
*                                                                           *
* Function:      Macro that returns a pointer to the front (current)        *
*                element in the parameter buffer. Used to obtain the        *
*                address of the next frame to be received/transmitted.      *
*                                                                           *
* Input:         buf_front(                                                 *
*                  buffer b; buffer structure (NOT a pointer to it).        *
*                );                                                         *
*                                                                           *
* Output:        Pointer to front (current) frame in the buffer.            *
*                                                                           *
* Globals:       None.                                                      *
*                                                                           *
* Edit History: 07/08/90 - Written by Robin T. Laird.                       *
*                                                                           *
\*****************************************************************************/

#define buf_front(b) (b.item[b.front])


/*****************************************************************************\
*                                  buf_rear                                   *
*****************************************************************************
*                                                                           *
* Function:      Macro that returns a pointer to the rear (last)            *
*                element in the parameter buffer. Used to obtain the        *
*                address of the next frame to be received/transmitted.      *
*                                                                           *
* Input:         buf_rear(                                                  *
*                  buffer b; buffer structure (NOT a pointer to it).        *
```



```
*                );                                                         *
*                                                                           *
* Output:        Pointer to rear (last) frame in the buffer.                *
*                                                                           *
* Globals:       None.                                                      *
*                                                                           *
* Edit History: 07/10/90 - Written by Robin T. Laird.                       *
*                                                                           *
\*****************************************************************************/

#define buf_rear(b) (b.item[b.rear])


/*****************************************************************************\
*                                  buf_inc                                    *
*****************************************************************************
*                                                                           *
* Function:      Macro that increments either the front or rear index       *
*                of a circular buffer of max size MAX_BUFFER_SIZE.          *
*                Should be used with care, i.e., not nested within a        *
*                compound statement like if ... then ... else.              *
*                                                                           *
* Input:         buf_inc(                                                   *
*                  word x; index to increment.                              *
*                );                                                         *
*                                                                           *
* Output:        Nothing.                                                   *
*                                                                           *
* Globals:       None.                                                      *
*                                                                           *
* Edit History: 12/04/90 - Written by Robin T. Laird.                       *
*                                                                           *
\*****************************************************************************/

#define buf_inc(x) (x = ((x == MAX_BUFFER_SIZE-1) ? 0 : x+1))


/*****************************************************************************\
*                                  buf_full                                   *
*****************************************************************************
*                                                                           *
* Function:      Function that returns the boolean of whether or not the    *
*                parameter buffer is full (TRUE if so, FALSE if not).       *
*                                                                           *
* Input:         buf_full(                                                  *
*                  buffer *b; pointer to buffer structure.                  *
*                );                                                         *
*                                                                           *
* Output:        Integer, TRUE if buffer FULL, FALSE if buffer not FULL.    *
*                                                                           *
* Globals:       None.                                                      *
*                                                                           *
* Edit History: 07/08/90 - Written by Robin T. Laird.                       *
*                                                                           *
\*****************************************************************************/

static int buf_full(b)
buffer *b;
{
  gcd_error = AOK;                      /* Assume function successful...   */
  return(b->full);
}


/*****************************************************************************\
*                                  buf_empty                                  *
*****************************************************************************
*                                                                           *
* Function:      Function that returns the boolean of whether or not the    *
*                parameter buffer is empty (TRUE if so, FALSE if not).      *
*                                                                           *
* Input:         buf_empty(                                                 *
*                  buffer *b; pointer to buffer structure.                  *
```

```
147   *                );                                                      *
148   *                                                                        *
149   * Output:        Integer, TRUE if buffer EMPTY, FALSE if buffer not EMPTY.*
150   *                                                                        *
151   * Globals:       None.                                                   *
152   *                                                                        *
153   * Edit History: 07/08/90 - Written by Robin T. Laird.                    *
154   *                                                                        *
155   \************************************************************************/
156
157   static int buf_empty(b)
158   buffer *b;
159   {
160     gcd_error = AOK;                        /* Assume function successful...    */
161     return(b->empty);
162   }
163
164
165   /************************************************************************\
166   *                                buf_clear                               *
167   ************************************************************************
168   *                                                                        *
169   * Function:      Initializes the parameter buffer and clears its contents.*
170   *                The front and rear pointers are reset and the boolean   *
171   *                state flags (i.e., empty, full) are set accordingly.    *
172   *                                                                        *
173   * Input:         buf_clear(                                              *
174   *                  buffer *b; pointer to buffer structure to clear.      *
175   *                );                                                      *
176   *                                                                        *
177   * Output:        Nothing.                                                *
178   *                                                                        *
179   * Globals:       gcd_error : module GCD.C                                *
180   *                                                                        *
181   * Edit History: 07/09/90 - Written by Robin T. Laird.                    *
182   *                                                                        *
183   \************************************************************************/
184
185   static void buf_clear(b)
186   buffer *b;
187   {
188     word i, j;
189
190     gcd_error = AOK;                        /* Assume function successful...    */
191
192     /* Process each of the frames in the buffer.                          */
193     /* Set all bytes in frame to 0.                                       */
194
195     for (i = 0; i < MAX_BUFFER_SIZE; i++)
196       for (j = 0; j < GCD_MAX_FRAME_LENGTH; j++) b->item[i][j] = 0x00;
197
198     /* Set the front and rear indexes equal to indicate empty buffer.     */
199     /* Set the empty and full flags as appropriate.                       */
200
201     b->front = b->rear = 0;
202     b->empty = TRUE;
203     b->full  = FALSE;
204   }
205
206
207   /************************************************************************\
208   *                                buf_insert                              *
209   ************************************************************************
210   *                                                                        *
211   * Function:      Inserts a frame into the parameter buffer if room.      *
212   *                The frame is not inserted if the source address equals  *
213   *                the destination - this avoids sending a frame to ones self.*
214   *                An error is returned if the buffer is already full.     *
215   *                                                                        *
216   * Input:         buf_insert(                                             *
217   *                  buffer *b;   pointer to buffer structure to insert into.*
218   *                  gcd_frame f; frame to be inserted.                    *
219   *                );                                                      *
```



```
220   *                                                                        *
221   * Output:        Nothing.                                                *
222   *                                                                        *
223   * Globals:       gcd_error : module GCD.C                                *
224   *                                                                        *
225   * Edit History: 07/09/90 - Written by Robin T. Laird.                    *
226   *               01/30/91 - Robin T. Laird added insertion of node ID into f.*
227   *               02/27/91 - Robin T. Laird added check for src equal dest.*
228   *                                                                        *
229   \************************************************************************/
230
231   static void buf_insert(b, f)
232   buffer *b;
233   gcd_frame f;
234   {
235     byte node_id;
236     word i, length;
237
238     gcd_error = AOK;                        /* Assume function successful...    */
239
240     /* Make sure buffer isn't already full.                               */
241
242     if (b->full)
243     {
244       gcd_error = ERR_FULL_BUFFER;
245       return;
246     }
247
248     /* Make sure we aren't trying to send a frame to ourself.             */
249     /* Insert source address at appropriate place in frame.               */
250     /* Source is obtained from node ID function at lowest level.          */
251
252     if ((node_id = gsc_node_id()) == f[GCD_DEST_ADDR_POS])
253     {
254       gcd_error = ERR_SRC_EQUAL_DEST;
255       return;
256     }
257     else
258       f[GCD_SRC_ADDR_POS] = node_id;
259
260     /* Extract length of frame from frame data (indexed at GCD_LEN_POS).  */
261     /* Copy element into buffer and set appropriate structure fields.     */
262     /* Length of frame is included as part of frame data (f[GCD_LEN_POS]).*/
263
264     length = f[GCD_LEN_POS];
265     for (i = 0; i < length; i++) b->item[b->rear][i] = f[i];
266
267     /* Adjust rear index (MAX_BUFFER_SIZE-1 is last element in buffer).   */
268
269     buf_inc(b->rear);
270
271     /* Check and see if we've filled the buffer (set full flag if so).    */
272     /* Set empty flag to FALSE since we just inserted.                    */
273
274     if ((b->front == b->rear) && !b->empty) b->full = TRUE;
275     b->empty = FALSE;
276   }
277
278
279   /************************************************************************\
280   *                                buf_remove                              *
281   ************************************************************************
282   *                                                                        *
283   * Function:      Removes a frame from the parameter buffer if available. *
284   *                An error is returned if the buffer is already empty.    *
285   *                                                                        *
286   * Input:         buf_remove(                                             *
287   *                  buffer *b;   pointer to buffer structure to remove from.*
288   *                  gcd_frame f; frame to be removed.                     *
289   *                );                                                      *
290   *                                                                        *
291   * Output:        Nothing.                                                *
292   *                                                                        *
```

```
293   * Globals:       gcd_error : module GCD.C                                *
294   *                                                                         *
295   * Edit History: 07/09/90 - Written by Robin T. Laird.                     *
296   *                                                                         *
297   \*************************************************************************/
298
299   static void buf_remove(b, f)
300   buffer *b;
301   gcd_frame f;
302   {
303     word i, length;
304
305     gcd_error = AOK;                        /* Assume function successful...    */
306
307     /* Make sure buffer isn't already empty.                                    */
308
309     if (b->empty)
310     {
311       gcd_error = ERR_EMPTY_BUFFER;
312       return;
313     }
314
315     /* Extract length of frame from frame data (indexed at GCD_LEN_POS).       */
316     /* Copy element into frame and set appropriate structure fields.           */
317     /* Buffer can't be full since we just removed an element.                   */
318
319     length = b->item[b->front][GCD_LEN_POS];
320     for (i = 0; i < length; i++) f[i] = b->item[b->front][i];
321     b->full = FALSE;
322
323     /* Adjust front index (MAX_BUFFER_SIZE-1 is last element in buffer).        */
324
325     buf_inc(b->front);
326
327     /* Check and see if we've depleted the buffer (set empty flag if so).      */
328
329     if (b->front == b->rear) b->empty = TRUE;
330   }
```

```
1   /*****************************************************************************\
2   *                                  GCD.H                                      *
3   ****************************************************************** ************
4   *                                                                             *
5   * CPCI:          IED90-MRA-ICN-GCS-GCD-H-R0C0                                 *
6   *                                                                             *
7   * Description:   GCS communications device handler variables and functions.   *
8   *                Contains constant function parameter declarations as well    *
9   *                as function return values (for success and failure of all    *
10  *                operations). Contains the function prototypes for the GCD.C  *
11  *                module.                                                      *
12  *                                                                             *
13  *                Module GCD exports the following types/variables/functions:  *
14  *                                                                             *
15  *                typedef gcd_frame;                                           *
16  *                typedef gcd_state;                                           *
17  *                                                                             *
18  *                int gcd_error;                                               *
19  *                                                                             *
20  *                gcd_init();                                                  *
21  *                gcd_reset();                                                 *
22  *                gcd_enable();                                                *
23  *                gcd_disable();                                               *
24  *                gcd_receive_frame();                                         *
25  *                gcd_transmit_frame();                                        *
26  *                gcd_status();                                                *
27  *                                                                             *
28  * Notes:         1) See SDS pp. 5-6 through 5-x for more information.         *
29  *                                                                             *
30  * Edit History: 06/29/90 - Written by Robin T. Laird.                         *
31  *                                                                             *
32  \*****************************************************************************/
33
34  /* Public Data Structures:                                                   */
35
36  #ifndef GCD_MODULE_CODE
37  #define GCD_MODULE_CODE          1000
38
39  #define GCD_ERR_NOT_INIT         1+GCD_MODULE_CODE
40  #define GCD_ERR_FRAME_LENGTH     2+GCD_MODULE_CODE
41  #define GCD_ERR_NUM_ATTEMPTS     3+GCD_MODULE_CODE
42
43  #define GCD_ERR_RECEIVE_FRAME    4+GCD_MODULE_CODE
44  #define GCD_ERR_TRANSMIT_FRAME   5+GCD_MODULE_CODE
45
46  #define GCD_FAIL_RECEIVER        6+GCD_MODULE_CODE
47  #define GCD_FAIL_TRANSMITTER     7+GCD_MODULE_CODE
48
49  #define GCD_WAIT_FOREVER         65535
50  #define GCD_DONT_WAIT            0
51
52  #define GCD_MAX_FRAME_LENGTH     SYS_MAX_PACKET_SIZE
53  #define GCD_MAX_ATTEMPTS         60000
54
55  #define GCD_DEST_ADDR_POS        0
56  #define GCD_LEN_POS              1
57  #define GCD_SRC_ADDR_POS         2
58
59  /* Type for receive/transmit data frame, simply a 256-element array.        */
60
61  typedef byte gcd_frame[GCD_MAX_FRAME_LENGTH];
62
63  /* Structure type for GCD module status (holds rcv/xmt error counts).       */
64
65  typedef struct { word r_valid_cnt;
66                   word r_err_cnt;
67                   word r_crce_err_cnt;
68                   word r_ae_err_cnt;
69                   word r_rcabt_err_cnt;
70                   word r_ovr_err_cnt;
71                   word x_valid_cnt;
72                   word x_err_cnt;
73                   word x_noack_err_cnt;
```



```
74                   word x_ur_err_cnt;
75                   word x_tcdt_err_cnt;
76                 } gcd_state;
77
78  /* External module global error variable.                                   */
79
80  extern int gcd_error;
81
82  /* Public Functions:                                                        */
83
84  void gcd_init(void);
85  void gcd_reset(void);
86  void gcd_enable(void);
87  void gcd_disable(void);
88  void gcd_receive_frame(gcd_frame f, word retry);
89  void gcd_transmit_frame(gcd_frame f, word retry);
90  void gcd_status(gcd_state *s);
91
92  #endif
```

```
/*****************************************************************************\
*                                  GCD.C                                      *
*****************************************************************************
*                                                                            *
* CPCI:          JED90-MRA-ICN-GCS-GCD-C-R0C1                                *
*                                                                            *
* Description:   GCS communications device handler functions.                *
*                Implements the MRA ICN standard Global Communications       *
*                Device (GCD) Handler module. This module contains the       *
*                standard device handler functions and must include the      *
*                low-level data link layer functions for the actual hardware *
*                implementation (currently implemented for the 80C152 GSC).  *
*                                                                            *
*                Message format at this level (ISO OSI data link layer) is:  *
*                                                                            *
*                 | byte 0 | byte 1 | byte 2 | byte 3 | byte 4 | byte n|     *
*                 |-------------------------------------------------|        *
*                 |  DEST  | LENGTH | SOURCE |  xxxx  |  xxxx  | .... |     *
*                                                                            *
*                Module GCD exports the following variables/functions:       *
*                                                                            *
*                int gcd_error;                                              *
*                                                                            *
*                gcd_init();                                                 *
*                gcd_reset();                                                *
*                gcd_enable();                                               *
*                gcd_disable();                                              *
*                gcd_receive_frame();                                        *
*                gcd_transmit_frame();                                       *
*                gcd_status();                                               *
*                                                                            *
* Notes:         1) The files BMF.H and BMF.C contain the support functions  *
*                   for managing the receive and transmit circular buffers.  *
*                2) The files GSC.H and GSC.C contain the required support   *
*                   functions for the Global Serial Channel hardware.        *
*                                                                            *
* Edit History: 06/29/90 - Written by Richard P. Smurlo and Robin T. Laird.  *
*                                                                            *
\*****************************************************************************/

#include <sysdefs.h>                    /* System constants and types.      */
#include "gcd.h"                        /* GCD public literals/functions.   */
#include "bmf.h"                        /* Buffer management functions.     */
#include "gsc.h"                        /* Global Serial Channel functions. */

#if defined(DEBUG)
#include <debug.h>                      /* Definitions for printf(), etc.   */
#endif

/* Public Variables:                                                         */

/* Global module error variable, gcd_error.                                 */
/* gcd_error contains code of last error occurence.                          */
/* Should be set to AOK after each successful function call.                 */
/* Variable can be examined by other software after each function call.      */

XDATA int gcd_error = GCD_ERR_NOT_INIT; /* Global module error variable.    */

/* Global module state variable, _gcd_state.                                 */
/* Holds GCD module error counts for frame reception/transmission.           */

static XDATA gcd_state _gcd_state;      /* Global module state variable.    */

/* Globals that tracks number of xmt/rcv attempts.                           */

static XDATA int gcd_tries = 0;         /* Number of xmt/rcv attempts.      */
static XDATA int gcd_times = 0;         /* Number of attempts to make.      */
static XDATA int gcd_stop  = FALSE;     /* Indicates when to stop trying.   */

/* Buffer management functions should be included here.                       */

#include "bmf.c"                        /* Buffer management functions.     */
```



```
/* Low-level data link layer support functions should be included here.     */

#include "gsc.c"                        /* Low-level GSC functions.         */


/*****************************************************************************\
*                                  gcd_init                                   *
*****************************************************************************
*                                                                            *
* Function:      Initializes the Global Communications Device Handler.       *
*                Clears the transmit and receive buffers, obtains the ICN    *
*                node ID, initializes the Global Serial Channel and the      *
*                associated transmit and receive DMA channels, and performs  *
*                a self-test of the GSC. The GSC reception/transmission      *
*                error and valid frame counters of the module state structure*
*                are cleared.                                                *
*                                                                            *
* Input:         gcd_init();                                                 *
*                                                                            *
* Output:        Nothing.                                                    *
*                                                                            *
* Globals:       gcd_error  : module GCD.C                                   *
*                _gcd_state : module GCD.C                                   *
*                rcv_buffer : module BMF.C                                   *
*                xmt_buffer : module BMF.C                                   *
*                                                                            *
* Edit History: 07/08/90 - Written by Robin T. Laird.                        *
*                                                                            *
\*****************************************************************************/

void gcd_init()
{
  gcd_error = AOK;                      /* Assume function successful...    */

  /* Reset all error counting registers in module state variable.           */
  /* Valid reception/transmission counters are also cleared.                */

  _gcd_state.r_valid_cnt     = 0;
  _gcd_state.r_err_cnt       = 0;
  _gcd_state.r_crce_err_cnt  = 0;
  _gcd_state.r_ae_err_cnt    = 0;
  _gcd_state.r_rcabt_err_cnt = 0;
  _gcd_state.r_ovr_err_cnt   = 0;
  _gcd_state.x_valid_cnt     = 0;
  _gcd_state.x_err_cnt       = 0;
  _gcd_state.x_noack_err_cnt = 0;
  _gcd_state.x_ur_err_cnt    = 0;
  _gcd_state.x_tcdt_err_cnt  = 0;

  /* Initialize the transmit and receive buffers.                           */

  buf_clear(&xmt_buffer);
  buf_clear(&rcv_buffer);

  /* Get the node ID for this particular ICN, and pass to GSC init routine. */
  /* No errors are currently fatal (they are only warnings).                */

  gsc_sys_init(gsc_node_id());
  if (gcd_error != AOK) return;

  /* Initialize the GSC DMA controllers.                                    */
  /* No errors are currently possible.                                      */

  gsc_dma_init();
  if (gcd_error != AOK) return;

  /* Set up the GSC DMA channel receive/transmit destination/source params. */
  /* The parameters include the DMA src/dest addresses and the byte count.  */
  /* Incoming frames go into the front of the rcv_buffer.                   */
  /* Outgoing frames go into the front of the xmt_buffer.                   */

  gsc_set_rcv_dst(buf_rear(rcv_buffer), GCD_MAX_FRAME_LENGTH);
  gsc_set_xmt_src(buf_front(xmt_buffer), GCD_MAX_FRAME_LENGTH);
```

```
147
148     /* Start the GSC DMA receive channel (allow DMA to function).          */
149     /* Start the GSC receiver (begin receiving data frames).               */
150     /* Start the GSC transmitter for hardware based acknowledges.          */
151     /* Errors are non-fatal and would cause only degraded performance.     */
152
153     gsc_rcv_go();
154     gsc_start_rcv();
155     if (gcd_error != AOK) return;
156     gsc_start_xmt();
157
158     /* Enable interrupts...                                                */
159
160     gsc_enable_interrupts();
161   }
162
163
164   /*************************************************************************\
165   *                              gcd_reset                                  *
166   *************************************************************************
167   *                                                                        *
168   * Function:     Performs a soft reset of the GCD systems.                *
169   *               The receive and transmit buffers are cleared a ; the     *
170   *               receive and transmit destination and source addresses    *
171   *               for the GSC DMA channels are reset to the beginning of   *
172   *               the buffers. The GSC reception/transmission error and    *
173   *               valid frame counters of the module state structure are   *
174   *               cleared.                                                 *
175   *                                                                        *
176   * Input:        gcd_reset();                                             *
177   *                                                                        *
178   * Output:       Nothing.                                                 *
179   *                                                                        *
180   * Globals:      gcd_error  : module GCD.C                                *
181   *               _gcd_state : module GCD.C                                *
182   *               rcv_buffer : module BMF.C                                *
183   *               xmt_buffer : module BMF.C                                *
184   *                                                                        *
185   * Edit History: 07/09/90 - Written by Robin T. Laird.                    *
186   *                                                                        *
187   \************************************************************************/
188
189   void gcd_reset()
190   {
191     gcd_error = AOK;                        /* Assume function successful...  */
192
193     /* Reset all error counting registers in module state variable.        */
194     /* Valid reception/transmission counters are also cleared.             */
195
196     _gcd_state.r_valid_cnt      = 0;
197     _gcd_state.r_err_cnt        = 0;
198     _gcd_state.r_crce_err_cnt   = 0;
199     _gcd_state.r_ae_err_cnt     = 0;
200     _gcd_state.r_rcabt_err_cnt  = 0;
201     _gcd_state.r_ovr_err_cnt    = 0;
202     _gcd_state.x_valid_cnt      = 0;
203     _gcd_state.x_err_cnt        = 0;
204     _gcd_state.x_noack_err_cnt  = 0;
205     _gcd_state.x_ur_err_cnt     = 0;
206     _gcd_state.x_tcdt_err_cnt   = 0;
207
208     /* Re-initialize the transmit and receive buffers.                     */
209
210     buf_clear(&xmt_buffer);
211     buf_clear(&rcv_buffer);
212
213     /* Set up the GSC DMA channel receive/transmit destination/source params. */
214     /* The parameters include the DMA src/dest addresses and the byte count.  */
215     /* Incoming frames go into the front of the rcv_buffer.                */
216     /* Outgoing frames go into the front of the xmt_buffer.                */
217
218     gsc_set_rcv_dst(buf_rear(rcv_buffer), GCD_MAX_FRAME_LENGTH);
219     gsc_set_xmt_src(buf_front(xmt_buffer), GCD_MAX_FRAME_LENGTH);
```



```
220   }
221
222
223   /*************************************************************************\
224   *                              gcd_enable                                 *
225   *************************************************************************
226   *                                                                        *
227   * Function:     Enables reception and transmission of data frames.       *
228   *               The GSC receiver and transmitter are re-enabled.         *
229   *               The function assumes that the GSC has been disabled      *
230   *               for some reason. It is normally not necessary to enable  *
231   *               the GSC after it has been initialized (by gcd_init()).   *
232   *                                                                        *
233   * Input:        gcd_enable();                                            *
234   *                                                                        *
235   * Output:       Nothing.                                                 *
236   *                                                                        *
237   * Globals:      gcd_error : module GCD.C                                 *
238   *                                                                        *
239   * Edit History: 07/09/90 - Written by Robin T. Laird.                    *
240   *                                                                        *
241   \************************************************************************/
242
243   void gcd_enable()
244   {
245     gcd_error = AOK;                        /* Assume function successful...  */
246
247     /* Start both the receiver and transmitter.                            */
248
249     gsc_start_rcv();
250     gsc_start_xmt();
251   }
252
253
254   /*************************************************************************\
255   *                              gcd_disable                                *
256   *************************************************************************
257   *                                                                        *
258   * Function:     Disables reception and transmission of data frames.      *
259   *               The GSC receiver and transmitter are disabled.           *
260   *               Any incoming data will be lost (dropped). The GSC must   *
261   *               be re-enabled using gcd_enable() before frames may be    *
262   *               received or transmitted again.                           *
263   *                                                                        *
264   * Input:        gcd_disable();                                           *
265   *                                                                        *
266   * Output:       Nothing.                                                 *
267   *                                                                        *
268   * Globals:      gcd_error : module GCD.C                                 *
269   *                                                                        *
270   * Edit History: 07/09/90 - Written by Robin T. Laird.                    *
271   *                                                                        *
272   \************************************************************************/
273
274   void gcd_disable()
275   {
276     gcd_error = AOK;                        /* Assume function successful...  */
277
278     /* Stop both the receiver and transmitter.                             */
279
280     gsc_stop_rcv();
281     gsc_stop_xmt();
282   }
283
284
285   /*************************************************************************\
286   *                           gcd_receive_frame                            *
287   *************************************************************************
288   *                                                                        *
289   * Function:     Removes a frame from the receive buffer if possible.     *
290   *               If a frame is available, it is removed from the receive  *
291   *               buffer, and returned in the parameter f. If a frame is not *
292   *               available from the receive buffer, the function will perform *
```

```
293   *                as follows depending upon the re-try value:                  *
294   *                                                                             *
295   *                GCD_DONT_WAIT     : remove frame if available, return if not.*
296   *                GCD_WAIT_FOREVER  : wait forever for frame to be received.   *
297   *                retry             : try this many times to receive frame.    *
298   *                                                                             *
299   * Input:         gcd_receive_frame(                                           *
300   *                  gcd_frame f; frame to be received (destination).           *
301   *                  word retry;  number of times to try receiving frame.       *
302   *                );                                                           *
303   *                                                                             *
304   * Output:        Nothing.                                                     *
305   *                                                                             *
306   * Globals:       gcd_error  : module GCD.C                                    *
307   *                gcd_tries  : module GCD.C                                    *
308   *                gcd_times  : module GCD.C                                    *
309   *                rcv_buffer : module BMF.C                                    *
310   *                                                                             *
311   * Edit History: 07/08/90 - Written by Robin T. Laird.                         *
312   *               01/30/91 - Robin T. Laird added re-receive count/stop.        *
313   *                                                                             *
314   \*****************************************************************************/
315
316   void gcd_receive_frame(f, retry)
317   gcd_frame f;
318   word retry;
319   {
320
321   #if defined(DEBUG)
322   unsigned char prev_rstat;       /* Last stored value of RSTAT.              */
323   unsigned char prev_bcrh0;       /* Last BCR's (high and low bytes).         */
324   unsigned char prev_bcrl0;
325   #endif
326
327     #if defined(DEBUG)
328     prev_rstat = 0xFF;
329     prev_bcrh0 = 0xFF;
330     prev_bcrl0 = 0xFF;
331     #endif
332
333     gcd_error = AOK;                          /* Assume function successful...  */
334
335     gcd_tries = 0;                            /* Number of attempted rcvs.      */
336     gcd_times = retry;                        /* Number of times to try.        */
337
338     /* If we don't want to wait (GCD_DONT_WAIT):                                */
339     /*   Try and remove frame from buffer.                                      */
340     /*   If one not available then gcd_error = ERR_BUF_EMPTY.                   */
341     /*   Else, we've successfully removed a received frame.                     */
342
343     /* If we want to wait forever (GCD_WAIT_FOREVER):                           */
344     /*   Wait until there is a frame in the buffer, then remove it.             */
345     /*   This function WILL wait forever, i.e., keep trying forever.            */
346
347     /* If we want to try a certain number of times:                             */
348     /*   Set up and zero auxilliary loo counter.                                */
349     /*   Loop, checking to see if frame is available and counter not expired.   */
350     /*   If counter expires, indicate buffer empty and return.                  */
351     /*   Else, get incoming frame and return.                                   */
352
353     if (retry == GCD_DONT_WAIT)
354     {
355       buf_remove(&rcv_buffer, f);
356     }
357     else if (retry == GCD_WAIT_FOREVER)
358     {
359       while (buf_empty(&rcv_buffer))
360
361         #if defined(DEBUG)
362         if ((prev_rstat != RSTAT)||(prev_bcrh0 != BCRH0)||(prev_bcrl0 != BCRL0))
363         {
364           printf("gcd_receive_frame: RSTAT=%02bx, BCR(HL)0=%02bx%02bx\n", RSTAT,
365           prev_rstat = RSTAT;
```



```
366           prev_bcrh0 = BCRH0;
367           prev_bcrl0 = BCRL0;
368         }
369         #endif
370         ;
371
372       buf_remove(&rcv_buffer, f);
373     }
374     else if (retry <= GCD_MAX_ATTEMPTS)
375     {
376       while (buf_empty(&rcv_buffer))
377         if (++gcd_tries >= gcd_times)
378         {
379           gcd_error = GCD_ERR_RECEIVE_FRAME;
380           break;
381         }
382     }
383     else
384     {
385       gcd_error = GCD_ERR_NUM_ATTEMPTS;
386     }
387   }
388
389
390   /*****************************************************************************\
391   *                              gcd_transmit_frame                             *
392   *****************************************************************************
393   *                                                                             *
394   * Function:      Adds the parameter frame to the transmit buffer if possible. *
395   *                If the transmit buffer is full an error is generated.        *
396   *                Otherwise, if the buffer is empty, the frame is transmitted  *
397   *                immediately, and the frame information is inserted into the  *
398   *                transmit buffer. Depending upon the re-try value, the        *
399   *                function will perform as follows:                            *
400   *                                                                             *
401   *                GCD_DONT_WAIT     : return immediately, frame added to buffer.*
402   *                GCD_WAIT_FOREVER  : wait forever for frame to be transmitted.*
403   *                retry             : try this many times to transmit frame.   *
404   *                                                                             *
405   *                Currently, only the GCD_WAIT_FOREVER option is supported.    *
406   *                This is equivalent to a one frame deep transmit buffer.      *
407   *                                                                             *
408   * Input:         gcd_transmit_frame(                                          *
409   *                  gcd_frame f; frame to be transmitted.                      *
410   *                  word retry;  number of times to try transmitting frame.    *
411   *                );                                                           *
412   *                                                                             *
413   * Output:        Nothing.                                                     *
414   *                                                                             *
415   * Globals:       gcd_error  : module GCD.C                                    *
416   *                _gcd_state : module GCD.C (DEBUG)                            *
417   *                gcd_tri    : module GCD.C                                    *
418   *                gcd_time   : module GCD.C                                    *
419   *                gcd_stop   : module GCD.C                                    *
420   *                xmt_buffer : module BMF.C                                    *
421   *                                                                             *
422   * Edit History: 07/08/90 - Written by Robin T. Laird.                         *
423   *               01/30/91 - Robin T. Laird added re-transmit count/stop.       *
424   *                                                                             *
425   \*****************************************************************************/
426
427   void gcd_transmit_frame(f, retry)
428   gcd_frame f;
429   word retry;
430   {
431
432   #if defined(DEBUG)
433   unsigned char   prev_tstat;     /* Last stored value of TSTAT.              */
434   unsigned char   prev_bcrh1;     /* Last BCR's (high and low bytes).         */
435   unsigned char   prev_bcrl1;
436   #endif
437
438     #if defined(DEBUG)
```

```
439     prev_tstat = 0xFF;
440     prev_bcrhl = 0xFF;
441     prev_bcrll = 0xFF;
442     #endif
443
444     gcd_error  = ACK;                       /* Assume function successful...   */
445
446     gcd_tries  = 0;                         /* Number of attempted xmts.       */
447     gcd_times  = retry;                     /* Number of times to try.         */
448     gcd_stop   = FALSE;                     /* Don't stop re-transmissions.    */
449
450     /* Insert frame into transmit buffer and send it.                          */
451     /* If the transmit buffer is empty then the GCS transmitter is disbaled:   */
452     /*   Reset the DMA transmitter source address and byte count.              */
453     /*   Re-start the GSC transmitter (it was turned off when buf empty).      */
454     /*   Re-enable the DMA transmit channel (also turned off when buf empty).  */
455     /* Else, just insert frame into buffer for transmission (sometime later).  */
456     /* If the insert failed, abort transmission and return an error.           */
457
458     if (buf_empty(&xmt_buffer))
459     {
460       buf_insert(&xmt_buffer, f);
461       if (gcd_error != AOK)
462       {
463         gcd_error = GCD_ERR_TRANSMIT_FRAME;
464         return;
465       }
466       else
467       {
468         gsc_set_xmt_src(buf_front(xmt_buffer), f[GCD_LEN_POS]);
469         gsc_start_xmt();
470         gsc_xmt_go();
471       }
472     }
473     else
474     {
475       buf_insert(&xmt_buffer, f);
476       if (gcd_error != AOK)
477       {
478         gcd_error = GCD_ERR_TRANSMIT_FRAME;
479         return;
480       }
481     }
482
483     /* At a later date, it will be possible to add a frame to the transmit     */
484     /* buffer and then return to the calling function immediately, the frame   */
485     /* would be transmitted when it moved to the front of the buffer.          */
486     /* A re-try of zero would indicate that the frame is to be transmitted     */
487     /* in the above manner (i.e., don't wait for frame to be transmitted).     */
488
489     if (retry == GCD_DONT_WAIT)
490     {
491       gcd_error = GCD_ERR_TRANSMIT_FRAME;
492     }
493     else if (retry == GCD_WAIT_FOREVER)
494     {
495       while (!buf_empty(&xmt_buffer))
496       {
497         #if defined(DEBUG)
498         if ((prev_tstat != TSTAT)||(prev_bcrhl != BCRHl)||(prev_bcrll != BCRLl))
499         {
500           printf("gcd_transmit_frame: TSTAT=%02bx, BCR(HL)l=%02bx%02bx\n", TSTAT,
501           prev_tstat = TSTAT;
502           prev_bcrhl = BCRHl;
503           prev_bcrll = BCRLl;
504         }
505         printf("noack = %u, ur = %u, tcdt = %u, xmt_valid = %u, rcv_valid = %u\n"
506           _gcd_state.x_noack_err_cnt,
507           _gcd_state.x_ur_err_cnt,
508           _gcd_state.x_tcdt_err_cnt,
509           _gcd_state.x_valid_cnt,
510           _gcd_state.r_valid_cnt);
511         #endif
```



```
512       }
513     }
514     else if (retry <= GCD_MAX_ATTEMPTS)
515     {
516       while (!buf_empty(&xmt_buffer))
517         if (gcd_stop)
518         {
519           buf_remove(&xmt_buffer, f);
520           gcd_error = GCD_ERR_TRANSMIT_FRAME;
521           break;
522         }
523     }
524     else
525     {
526       gcd_error = GCD_ERR_NUM_ATTEMPTS;
527     }
528   }
529
530
531   /*****************************************************************************\
532   *                                  gcd_status                                 *
533   *****************************************************************************
534   *                                                                            *
535   * Function:     Returns the operational status of the GCD subsystem.         *
536   *               This is accomplished by returning the contents of the        *
537   *               module state structure that records various operational      *
538   *               parameters such as the number of valid frames received, etc. *
539   *                                                                            *
540   * Input:        gcd_status(                                                  *
541   *                 gcd_state *s; pointer to module state structure.           *
542   *               );                                                           *
543   *                                                                            *
544   * Output:       Nothing.                                                     *
545   *                                                                            *
546   * Globals:      gcd_error : module GCD.C                                     *
547   *                                                                            *
548   * Edit History: 07/09/90 - Written by Richard P. Smurlo.                     *
549   *                                                                            *
550   \*****************************************************************************/
551
552   void gcd_status(s)
553   gcd_state *s;
554   {
555     gcd_error = AOK;                      /* Assume function successful...   */
556
557     *s = _gcd_state;
558   }
```

```
1   /*****************************************************************************\
2   *                                    GCI.H                                    *
3   *****************************************************************************
4   *                                                                             *
5   * CPCI:          IED90-MRA-ICN-GCS-GCI-H-R0C0                                 *
6   *                                                                             *
7   * Description:   GCS communications interface variables and functions.        *
8   *                Contains constant function parameter declarations as well    *
9   *                as function return values (for success and failure of all    *
10  *                operations). Contains the function prototypes for the GCI.C  *
11  *                module.                                                      *
12  *                                                                             *
13  *                Module GCI exports the following types/variables/functions:  *
14  *                                                                             *
15  *                typedef gci_message;                                         *
16  *                                                                             *
17  *                int gci_error;                                               *
18  *                                                                             *
19  *                gci_init();                                                  *
20  *                gci_receive_message();                                       *
21  *                gci_send_message();                                          *
22  *                                                                             *
23  * Notes:         1) See SDS pp. 5-6 through 5-x for more information.         *
24  *                                                                             *
25  * Edit History: 06/29/90 - Written by Robin T. Laird.                         *
26  *                                                                             *
27  \*****************************************************************************/
28
29  /* Public Data Structures:                                                   */
30
31  #ifndef GCI_MODULE_CODE
32  #define GCI_MODULE_CODE          2000
33
34  #define GCI_ERR_NOT_INIT         1+GCI_MODULE_CODE
35  #define GCI_ERR_RECEIVE_MESSAGE  2+GCI_MODULE_CODE
36  #define GCI_ERR_SEND_MESSAGE     3+GCI_MODULE_CODE
37
38  /* Maximum and minimum retry values for send/receive of packets.            */
39  /* Values must correspond with related definitions in module GCD.H.         */
40
41  #define GCI_WAIT_FOREVER         65535
42  #define GCI_DONT_WAIT            0
43  #define GCI_MAX_ATTEMPTS         60000
44
45  /* Maximum message length SHOULD be two less than maximum frame length.     */
46
47  #define GCI_MAX_MESSAGE_LENGTH   SYS_MAX_PACKET_SIZE
48
49  /* MRA inter-module message type defined as a sequence of bytes.            */
50
51  typedef byte gci_message[GCI_MAX_MESSAGE_LENGTH];
52
53  /* External module global error variable.                                   */
54
55  extern int gci_error;
56
57  /* Public Functions:                                                        */
58
59  void gci_init();
60  void gci_receive_message(gci_message m, word retry);
61  void gci_send_message(gci_message m, word retry);
62
63  #endif
```

```
/*****************************************************************************\
*                                   GCI.C                                     *
*****************************************************************************
*                                                                           *
* CPCI:          IED90-MRA-ICN-GCS-GCI-C-R0C0                               *
*                                                                           *
* Description:   GCS communications interface functions.                    *
*                Implements the MRA ICN standard Global Communications      *
*                Interface (GCI) module. This module contains the standard  *
*                communications interface functions that provide higher-level *
*                software access to the Intelligent Communications Node (ICN) *
*                Local Area Network (LAN).                                  *
*                                                                           *
*                Message format at this level (ISO OSI network layer) is:   *
*                                                                           *
*                 | byte 0 | byte 1 | byte 2 | byte 3 | byte 4 | byte n|     *
*                 |-------------------------------------------------|       *
*                 |  DEST  | LENGTH | SOURCE |  xxxx  |  xxxx  |  .... |     *
*                                                                           *
*                Module GCI exports the following variables/functions:      *
*                                                                           *
*                int gci_error;                                             *
*                                                                           *
*                gci_init();                                                *
*                gci_receive_message();                                     *
*                gci_send_message();                                        *
*                                                                           *
* Notes:         1) The GCI functions are implementation independent.       *
*                2) Module GCI represents the MRA Communications Level.     *
*                                                                           *
* Edit History: 07/28/90 - Written by Robin T. Laird.                       *
*                                                                           *
\*****************************************************************************/

#include <sysdefs.h>                    /* System constants and types.      */
#include "gcd.h"                        /* GCD public literals/functions.   */
#include "gci.h"                        /* GCI public literals/functions.   */

/* Public Variables:                                                        */

/* Global module error variable, gci_error.                                */
/* gci_error contains code of last error occurence.                         */
/* Should be set to AOK after each successful function call.                */
/* Variable can be examined by other software after each function call.     */

XDATA int gci_error = GCI_ERR_NOT_INIT; /* Global module error variable.    */


/*****************************************************************************\
*                                  gci_init                                   *
*****************************************************************************
*                                                                           *
* Function:      Initializes the Global Communications Interface.           *
*                The Global Communications Devic Handler (GCD) subsystem    *
*                along with all module variables are initialized. Any errors *
*                are examined for severity and an attempt is made to recover *
*                from non-fatal conditions. If initialization is unsuccessful *
*                then the error GCI_ERR_NOT_INIT is returned in gci_error.  *
*                                                                           *
* Input:         gci_init();                                                *
*                                                                           *
* Output:        Nothing.                                                   *
*                                                                           *
* Globals:       gci_error : module GCI.C                                   *
*                gcd_error : module GCD.C                                   *
*                                                                           *
* Edit History: 07/28/90 - Written by Robin T. Laird.                       *
*                                                                           *
\*****************************************************************************/

void gci_init()
{
  gci_error = AOK;                      /* Assume function successful...    */
```



```

  /* Initialize the GCD subsystem (sets up ICN LAN hardware and software).  */

  gcd_init();
  if (gcd_error != AOK)
  {
    switch(gcd_error)
    {
      case GCD_FAIL_RECEIVER:
      case GCD_FAIL_TRANSMITTER:
        gci_error = GCI_ERR_NOT_INIT;
        break;
      default:
        gci_error = gcd_error;
    }
  }
}


/*****************************************************************************\
*                             gci_receive_message                             *
*****************************************************************************
*                                                                           *
* Function:      Receives the latest (oldest) message from the ICN LAN.     *
*                If a message is not immediately available, the function    *
*                attempts a specific number of recpetions (as given below)  *
*                and then returns regardless. If a message is never received, *
*                then the global variable gci_error is set to the literal   *
*                GCI_ERR_RECEIVE_MESSAGE. If a message is available then it *
*                is returned immediately.                                   *
*                                                                           *
*                The number of receive errors is tracked so that if it      *
*                exceeds a maximum value over time, the GCS device handler  *
*                is reset to try and remedy the problem.                    *
*                                                                           *
* Input:         gci_receive_message(                                       *
*                  gci_message m; received message.                         *
*                  word retry;    number of times to try receiving message. *
*                );                                                         *
*                                                                           *
* Output:        Nothing.                                                   *
*                                                                           *
* Globals:       gci_error : module GCI.C                                   *
*                gcd_error : module GCD.C                                   *
*                                                                           *
* Edit History: 08/11/90 - Written by Robin T. Laird.                       *
*                                                                           *
\*****************************************************************************/

#define MAX_RECV_ERRORS         1000    /* On > 1000 errors, reset GCD.     */

void gci_receive_message(m, retry)
gci_message m;
word retry;
{
  gcd_state status;                     /* Holds GCD status.                */

  gci_error = AOK;                      /* Assume function successful...    */

  /* Request frame. Iterate retry number of times.                          */

  gcd_receive_frame(m, retry);

  /* If a frame is not available, check integrity of receiver.              */
  /* If number of receive errors is excessive then reset the GCD subsystem. */
  /* Currently only counting CRC and alignment errors.                      */

  /* If a frame is available then return it.                                */

  if (gcd_error != AOK)
  {
    gcd_status(&status);
    if (status.r_crce_err_cnt+status.r_ae_err_cnt > MAX_RECV_ERRORS) gcd_reset(
```

```
147       gci_error = GCI_ERR_RECEIVE_MESSAGE;
148     }
149   }
150
151
152   /*************************************************************************\
153   *                              gci_send_message                           *
154   *************************************************************************
155   *                                                                        *
156   * Function:     Sends the parameter message to the ICN LAN.              *
157   *               If the message cannot be sent immediately, the function  *
158   *               attempts a specific number of re-transmissions (given below) *
159   *               then returns regardless. If the message is never sent, then  *
160   *               the global variable gci_error is set to GCI_ERR_SEND_MESSAGE.*
161   *                                                                        *
162   *               The number of send errors is tracked so that if it       *
163   *               exceeds a maximum value over time, the GCS device handler *
164   *               is reset to try and remedy the problem.                  *
165   *                                                                        *
166   * Input:        gci_send_message(                                        *
167   *                 gci_message m; message to be sent.                     *
168   *                 word retry;    number of times to try sending message. *
169   *               );                                                       *
170   *                                                                        *
171   * Output:       Nothing.                                                 *
172   *                                                                        *
173   * Globals:      gci_error : module GCI.C                                 *
174   *               gcd_error : module GCD.C                                 *
175   *                                                                        *
176   * Edit History: 08/11/90 - Written by Robin T. Laird.                    *
177   *                                                                        *
178   \*************************************************************************/
179
180   #define MAX_SEND_ERRORS         1000    /* > 1000 errors, reset GCD.        */
181
182   void gci_send_message(m, retry)
183   gci_message m;
184   word retry;
185   {
186     gcd_state status;                     /* Holds GCD status.                */
187
188     gci_error = AOK;                      /* Assume function successful...    */
189
190     /* Send frame. Iterate retry number of times.                             */
191
192     gcd_transmit_frame(m, retry);
193
194     /* If frame could not be transmitted, check integrity of transmitter.     */
195     /* If number of transmit errors is excessive then reset the GCD subsystem.*/
196     /* Currently only counting no acknowledgement errors.                     */
197
198     if (gcd_error != AOK)
199     {
200       gcd_status(&status);
201       if (status.x_noack_err_cnt > MAX_SEND_ERRORS) gcd_reset();
202       gci_error = GCI_ERR_SEND_MESSAGE;
203     }
204   }
```

```
1   /*****************************************************************************\
2   *                                  GSC.H                                      *
3   *****************************************************************************
4   *                                                                           *
5   * CPCI:          IED90-MRA-ICN-GCS-GSC-H-R0C0                               *
6   *                                                                           *
7   * Description:   Global Serial Channel (GSC) variables and functions.       *
8   *                Contains constant function parameter and hardware register *
9   *                declarations for initializatin and control of the Intel    *
10  *                80C152 GSC. Implements the low-level data link layer portion *
11  *                of the MRA GCS communications device handler functions.    *
12  *                                                                           *
13  *                Module GSC exports the following functions:                *
14  *                                                                           *
15  *                gsc_node_id();                                             *
16  *                gsc_sys_init();                                            *
17  *                gsc_dma_init();                                            *
18  *                gsc_set_rcv_dst();                                         *
19  *                gsc_set_xmt_src();                                         *
20  *                gsc_start_rcv();                                           *
21  *                gsc_start_xmt();                                           *
22  *                gsc_rcv_go();                                              *
23  *                gsc_xmt_go();                                              *
24  *                gsc_stop_rcv();                                            *
25  *                gsc_stop_xmt();                                            *
26  *                gsc_enable_interrupts();                                   *
27  *                gsc_rcv_valid() interrupt;                                 *
28  *                gsc_xmt_valid() interrupt;                                 *
29  *                gsc_rcv_error() interrupt;                                 *
30  *                gsc_xmt_error() interrupt;                                 *
31  *                                                                           *
32  * Notes:         1) See the Intel 8-Bit Embedded Controller Handbook for    *
33  *                   more information (No. 270645-002, pp. 9-1 - 9-87).      *
34  *                                                                           *
35  * Edit History: 06/28/90 - Written by Richard P. Smurlo.                    *
36  *                                                                           *
37  \*****************************************************************************/
38
39  /* Private Data Structures:                                                */
40
41  #ifndef GSC_MODULE_CODE
42  #define GSC_MODULE_CODE          1200
43
44  #define ERR_NODE_ID_TOO_LARGE    1+GSC_MODULE_CODE
45  #define ERR_RFIFO_NOT_CLEAR      2+GSC_MODULE_CODE
46
47  /* GSC Constants:                                                         */
48
49  #define ADR_TFIFO           0x85 /* Transmit FIFO Address.                 */
50  #define ADR_RFIFO           0xF4 /* Receive FIFO Address.                  */
51  #define GSC_BAUD            0x01 /* GSC Baud Rate = (OSCf)/((GSC_BAUD+1)*8). */
52                                   /*   GSC_BAUD = 1 > Baud Rate = 921.6 KBPS. */
53  #define CLOCK_MASK          0x00 /* GMOD = 0b0XXX|XXXX for Internal clock.  */
54  #define D_BKOFF_GMOD        0x60 /* GMOD = 0bX11X|XXXX for alternate BKOFF. */
55  #define ADDR_MASK           0x00 /* GMOD = 0bXXX0|XXXX for 8-bit Addresses. */
56  #define CRC_MASK            0x00 /* GMOD = 0bXXXX|0XXX for 16-bit CRC.      */
57  #define PREAMBLE_MASK       0x02 /* GMOD = 0bXXXX|X01X for 8-bit preamble.  */
58  #define PROTOCOL_MASK       0x00 /* GMOD = 0bXXXX|XXX0 for CSMA/CD protocol. */
59
60  #define JAM_MASK            0x80 /* MYSLOT = 0b1XXX|XXXX for DC Jam Signal. */
61  #define D_BKOFF_SLOT        0x40 /* MYSLOT = 0bX1XX|XXXX for deterministic  */
62                                   /*   resolution.                           */
63  #define GSC_IFS             0    /* IFS = Number of Machine Cycles for      */
64                                   /*   longest Receive service routine       */
65  #define GSC_SLOTTM          216  /* 256-SLOTTM = # of bit times for round   */
66                                   /*   trip propagation and CRC jam time.    */
67  #ifdef ICNMON
68  #define GSC_HABEN           0    /* ICN Monitor should not Acknowledge.     */
69  #define GSC_AMSK0           0xFF /* AMSK0 = 0xFF - Mask off ADR0 Node ID.   */
70  #else
71  #define GSC_HABEN           1    /* HABEN = 1 for Hardware Based Acknowledges*/
72  #define GSC_AMSK0           0x00 /* AMSK0 = 0x00 - Don't mask ADR0 Node ID. */
73  #endif
```



```
74
75  #define GSC_AMSK1           0x00 /* AMSK1 = 0x00 - Don't mask off any part  */
76                                   /*   of Address 1.                         */
77  #define GSC_DMA             1    /* DMA = 1 for DMA control of GSC.         */
78
79  #define NO_SLOT             0x00 /* MYSLOT = 0bXX00|0000 if Node ID > 32.   */
80  #define MAX_SLOTS           32   /* Maximum number of slots used.           */
81
82  #define R_OVR_ERR_MASK      0x80 /* Receive over-run Error mask.            */
83  #define R_RCABT_ERR_MASK    0x40 /* Receive Abort Error mask.               */
84  #define R_AE_ERR_MASK       0x20 /* Receive Alignment Error mask.           */
85  #define R_CRCE_ERR_MASK     0x10 /* Receive CRC Error mask.                 */
86  #define X_NOACK_ERR_MASK    0x40 /* Transmit No Acknowledge Error mask.     */
87  #define X_UR_ERR_MASK       0x20 /* Transmit Under-run Error mask.          */
88  #define X_TCDT_ERR_MASK     0x10 /* Transmit Collision Detect Error mask.   */
89
90  /* DMA Constants:                                                         */
91
92  #define HLDHLDA_DISABLE     0x9F /* PCON = 0bX00X|XXXX to disable the Hold/ */
93                                   /*   Hold Acknowledge Logic.               */
94  #define R_XFERMODE_MASK     0x00 /* DCON0 = 0bXXXX|X0XX for response to GSC */
95                                   /*   interrupts, alternate cycle mode.     */
96  #define R_DMNDMODE_MASK     0x08 /* DCON0 = 0bXXXX|1XXX to enable demand mode*/
97  #define R_SRC_AUTOINC_MASK  0x00 /* DCON0 = 0bXXX0|XXXX to clear auto incre- */
98                                   /*   ment option for source addr.          */
99  #define R_SOURCE_MASK       0x20 /* DCON0 = 0bXX1X|XXXX to define DMA source */
100                                  /*   as SFR.                               */
101 #define R_DST_AUTOINC_MASK  0x40 /* DCON0 = 0bX1XX|XXXX to set auto increment*/
102                                  /*   option for receive dest. address.     */
103 #define R_DEST_MASK         0x00 /* DCON0 = 0b0XXX|XXXX for DMA dest as ex- */
104                                  /*   ternal memory.                        */
105
106 #define X_XFERMODE_MASK     0x00 /* DCON1 = 0bXXXX|X0XX for response to GSC */
107                                  /*   interrupts, alternate cycle mode.     */
108 #define X_DMNDMODE_MASK     0x08 /* DCON1 = 0bXXXX|1XXX to enable demand mode*/
109 #define X_SRC_AUTOINC_MASK  0x10 /* DCON1 = 0bXXX1|XXXX to auto increment   */
110                                  /*   source address.                       */
111 #define X_SOURCE_MASK       0x00 /* DCON1 = 0bXX0X|XXXX to select External  */
112                                  /*   RAM as source.                        */
113 #define X_DST_AUTOINC_MASK  0x00 /* DCON1 = 0bX0XX|XXXX to clear auto incre- */
114                                  /*   ment option for dest. addr.           */
115 #define X_DEST_MASK         0x80 /* DCON1 = 0b1XXX|XXXX to define DMA dest. */
116                                  /*   as SFR.                               */
117
118 /* Private Functions:                                                     */
119
120 static byte gsc_node_id(void);
121 static void gsc_sys_init(byte node_id);
122 static void gsc_dma_init(void);
123 static void gsc_set_rcv_dst(gcd_frame f, word length);
124 static void gsc_set_xmt_src(gcd_frame f, word length);
125 static void gsc_start_rcv(void);
126 static void gsc_start_xmt(void);
127 static void gsc_rcv_go(void);
128 static void gsc_xmt_go(void);
129 static void gsc_stop_rcv(void);
130 static void gsc_stop_xmt(void);
131 static void gsc_stop_xmt(void);
132 static void gsc_enable_interrupts();
133
134 #endif
```

```
/*****************************************************************************\
*                                   GSC.C                                     *
*****************************************************************************
*                                                                           *
* CPCI:          JED90-MRA-ICN-GCS-GSC-C-R0C2                               *
*                                                                           *
* Description:   Global Serial Channel (GSC) functions.                     *
*                Implements the MRA ICN low-level data link layer functions *
*                for the GCS global communications device handler.          *
*                Currently supports the 80C152 Global Serial Channel (GSC). *
*                                                                           *
*                Module GSC exports the following functions:                *
*                                                                           *
*                xptr_lo_offset() macro;                                    *
*                xptr_hi_offset() macro;                                    *
*                gsc_node_id();                                             *
*                gsc_sys_init();                                            *
*                gsc_dma_init();                                            *
*                gsc_set_rcv_dst();                                         *
*                gsc_set_xmt_src();                                         *
*                gsc_start_rcv();                                           *
*                gsc_start_xmt();                                           *
*                gsc_rcv_go();                                              *
*                gsc_xmt_go();                                              *
*                gsc_stop_rcv();                                            *
*                gsc_stop_xmt();                                            *
*                gsc_enable_interrupts();                                   *
*                gsc_rcv_valid() interrupt;                                 *
*                gsc_xmt_valid() interrupt;                                 *
*                gsc_rcv_error() interrupt;                                 *
*                gsc_xmt_error() interrupt;                                 *
*                                                                           *
* Notes:         1) This module is NOT a stand-alone compilation unit.      *
*                   It is included by the module GCD.C and is compiled there.*
*                   It is assumed that the file GCD.H is included before it.*
*                   Note that all of the functions herein are static.       *
*                                                                           *
* Edit History: 07/10/90 - Written by Richard P. Smurlo and Robin T. Laird. *
*                                                                           *
\*****************************************************************************/

#include <reg152.h>                     /* 8031 and 80152 definitions.      */


/*****************************************************************************\
*                                xptr_lo_offset                             *
*****************************************************************************
*                                                                           *
* Function:      Obtains the low-offset portion of an external data (XDATA) *
*                pointer. Specific to the Franklin 8031 C compiler (V2.4).  *
*                                                                           *
* Input:         xptr_lo_offset(                                            *
*                  xdata void *xp; XDATA pointer.                           *
*                );                                                         *
*                                                                           *
* Output:        Returns the low-offset portion of the external data pointer.*
*                                                                           *
* Globals:       None.                                                      *
*                                                                           *
* Edit History: 07/07/90 - Written by Robin T. Laird.                       *
*                                                                           *
\*****************************************************************************/

#define xptr_lo_offset(xp) (((unsigned int)xp)&0x00FF)


/*****************************************************************************\
*                                xptr_hi_offset                             *
*****************************************************************************
*                                                                           *
* Function:      Obtains the hi-offset portion of an external data (XDATA)  *
*                pointer. Specific to the Franklin 8031 C compiler (V2.4).  *
*                                                                           *
```



```
* Input:         xptr_hi_offset(                                            *
*                  xdata void *xp; XDATA pointer.                           *
*                );                                                         *
*                                                                           *
* Output:        Returns the hi-offset portion of the external data pointer.*
*                                                                           *
* Globals:       None.                                                      *
*                                                                           *
* Edit History: 07/07/90 - Written by Robin T. Laird.                       *
*                                                                           *
\*****************************************************************************/

#define xptr_hi_offset(xp) ((((unsigned int)xp)>>8)&0x00FF)


/*****************************************************************************\
*                                gsc_node_id                                *
*****************************************************************************
*                                                                           *
* Function:      Determines node ID number for the particular ICN.          *
*                The ID number is currently read from the P6 80C152         *
*                parallel port which is connected to an 8-position switch.  *
*                                                                           *
* Input:         gsc_node_id();                                             *
*                                                                           *
* Output:        Returns the node identification number (address) as a byte.*
*                Valid range is 0 <= node ID <= 0xFF.                       *
*                                                                           *
* Globals:       gcd_error   : module GCD.C                                 *
*                80C152 regs : module REG152.H                              *
*                                                                           *
* Edit History: 06/28/90 - Written by Robin T. Laird.                       *
*                                                                           *
\*****************************************************************************/

static byte gsc_node_id()
{
  gcd_error = AOK;                      /* Assume function successful...    */

  return ((byte)P6);
}


/*****************************************************************************\
*                                gsc_sys_init                               *
*****************************************************************************
*                                                                           *
* Function:      Initializes the Intel 80C152 Global Serial Channel (GSC).  *
*                The GSC is set up for operation as follows:                *
*                                                                           *
*                BAUD RATE        = 1.8432 MBPS                             *
*                CLOCK SOURCE     = internal baud rate generator            *
*                PROTOCOL         = CSMA/CD                                 *
*                PREAMBLE LENGTH  = 8-bits                                  *
*                BACKOFF MODE     = deterministic                           *
*                CRC LENGTH       = 16-bits                                 *
*                IFS              = 256 B/Ts                                *
*                JAMMING SIGNAL   = D.C.                                    *
*                SLOT TIME        = 40 B/Ts                                 *
*                ADDRESSING MODE  = 8-bits                                  *
*                HBA              = enabled                                 *
*                                                                           *
*                Deterministic collision resolution requires that slot      *
*                numbers be used, and limits this implementation to 64 slots*
*                or ICN nodes. Since hardware based acknowledge is being used*
*                we are also forcing use of even addresses. We have chosen  *
*                to base the slot number on the address number, which gives *
*                only 32 possible unique 8-bit addresses. If the node ID is *
*                greater than 32, then the GSC slot number is set to 0 and  *
*                a warning is signaled (ERR_NODE_ID_TOO_LARGE).             *
*                                                                           *
* Input:         gsc_sys_init(                                              *
*                  byte node_id; node address (which equals slot number).   *
```

```
147   *                );                                                    *
148   *                                                                      *
149   * Output:        Nothing.                                              *
150   *                                                                      *
151   * Globals:       gcd_error    : module GCD.C                           *
152   *                80C152 regs : module REG152.H                         *
153   *                                                                      *
154   * Edit History: 07/10/90 - Written by Richard P. Smurlo.               *
155   *                03/28/91 - Modified by Robin T. Laird added EA = 0.   *
156   *                                                                      *
157   \**********************************************************************/
158
159   static void gsc_sys_init(node_id)
160   byte node_id;
161   {
162     gcd_error = AOK;                        /* Assume function successful...    */
163
164     /* Disable interrupts while we're changing things.                          */
165
166     EA = 0;
167
168     /* Initialize all GSC related Registers to 0x00.                            */
169
170     GMOD    = 0x00;
171     MYSLOT  = 0x00;
172
173     /* Protocol Dependent Initialization.                                       */
174
175     BAUD    = GSC_BAUD;                     /* GSC Baud Rate Select.            */
176     GMOD    = GMOD   | PROTOCOL_MASK;       /* CSMA/CD or SDLC Protocol.        */
177     GMOD    = GMOD   | CLOCK_MASK;          /* Int. or Ext. clock.              */
178     GMOD    = GMOD   | PREAMBLE_MASK;       /* Message Preamble.                */
179     GMOD    = GMOD   | D_BKOFF_GMOD;        /* Deterministic Backoff.           */
180     MYSLOT  = MYSLOT | D_BKOFF_SLOT;        /* Ditto                            */
181     GMOD    = GMOD   | CRC_MASK;            /* CRC definition.                  */
182     IFS     = GSC_IFS;                      /* Inter Frame Space.(LSBit ignored)*/
183     MYSLOT  = MYSLOT | JAM_MASK;            /* Jam Signal Mask.                 */
184     SLOTTM  = GSC_SLOTTM;                   /* Slot Time.                       */
185     GMOD    = GMOD   | ADDR_MASK;           /* Addressing.                      */
186
187     /* The address mask registers are explicitly initialized to values that     */
188     /* are "safe" to gaurd against false recognition of data frames. Only       */
189     /* addresses that equal the node_id will be recognized. An un-advertised    */
190     /* feature is that a node will actually recognize the node_id address       */
191     /* (which is supposed to be an even value) AND it will recognize the        */
192     /* address that is one greater than the node_id. The odd address is used    */
193     /* when the sender does not want the HBA to be used.                        */
194
195     ADR0    = node_id;                      /* Addressing. (Node ID # 1)        */
196     ADR1    = node_id+1;                    /* Addressing. (Node ID # 2)        */
197     ADR2    = node_id;                      /* Addressing. (Node ID # 1)        */
198     ADR3    = node_id+1;                    /* Addressing. (Node ID # 2)        */
199     AMSK0   = GSC_AMSK0;                    /* AMSK0 masks ADR0 for ICNMON (FF).*/
200     AMSK1   = GSC_AMSK1;                    /* AMSK1 doesn't mask anything (00).*/
201     HABEN   = GSC_HABEN;                    /* Hardware Based Ack.              */
202
203     /* Deterministic Resolution Initialization.                                 */
204
205     TCDCNT  = MAX_SLOTS;                    /* Maximum number of slots.         */
206     PRBS    = 0xFF;                         /* Disable the PRBS for deter-      */
207                                             /*   ministic resolution.           */
208
209     /* Set up slot number of a given node. Higher number = higher priority.     */
210     /* Slot numbers are used in deterministic collision resolution.             */
211     /* Slot numbers will be based upon (equal to) node ID (address).            */
212     /* Currently, an error is flagged if the node_id > MAX_SLOTS.               */
213
214     if (node_id <= MAX_SLOTS)
215       MYSLOT = MYSLOT | node_id;
216     else
217     {
218       MYSLOT = MYSLOT | NO_SLOT;            /* Slot number 0 if Node ID > MAX.  */
219       gcd_error = ERR_NODE_ID_TOO_LARGE;    /* Warn that node ID is too big.    */
```



```
220     }
221
222     #if defined(DEBUG)
223     printf("gsc_sys_init: GMOD   = %02bx\n", GMOD);
224     printf("gsc_sys_init: BAUD   = <can't read>");
225     printf("gsc_sys_init: MYSLOT= %02bx\n", MYSLOT);
226     printf("gsc_sys_init: IFS    = %02bx (LSB indeterminate)\n", IFS);
227     printf("gsc_sys_init: ADR0   = %02bx\n", ADR0);
228     printf("gsc_sys_init: ADR1   = %02bx\n", ADR1);
229     printf("gsc_sys_init: ADR2   = %02bx\n", ADR2);
230     printf("gsc_sys_init: ADR3   = %02bx\n", ADR3);
231     printf("gsc_sys_init: SLOTTM= %02bx\n", SLOTTM);
232     printf("gsc_sys_init: AMSK0 = %02bx\n", AMSK0);
233     printf("gsc_sys_init: AMSK1 = %02bx\n", AMSK1);
234     printf("gsc_sys_init: RSTAT = %02bx\n", RSTAT);
235     printf("gsc_sys_init: TCDCNT= %02bx\n", TCDCNT);
236     printf("gsc_sys_init: PRBS   = %02bx\n", PRBS);
237     #endif
238   }
239
240
241   /**********************************************************************\
242   *                              gsc_dma_init                            *
243   ************************************************************************
244   *                                                                      *
245   * Function:      Initializes the Intel 80C152 Global Serial Channel (GSC)  *
246   *                DMA controller. DMA channel 0 is used for receiving frames *
247   *                while DMA channel 1 is used for transmitting frames.  *
248   *                The DMA channels are initialized as follows:          *
249   *                                                                      *
250   *                DMA channel 0 (receiver):                             *
251   *                                                                      *
252   *                TRANSFER MODE    = GSC interrupts used to initiate transfer *
253   *                DMA CHANNEL MODE = demand mode                        *
254   *                INCREMENT SOURCE = no                                 *
255   *                SOURCE ADDRESS   = RFIFO SFR                          *
256   *                INCREMENT DEST   = yes          (done in receive routine) *
257   *                DEST ADDRESS     = rcv_buffer[] (done in receive routine) *
258   *                                                                      *
259   *                DMA channel 1 (transmitter):                          *
260   *                                                                      *
261   *                TRANSFER MODE    = GSC interrupts used to initiate transfer *
262   *                DMA CHANNEL MODE = demand mode                        *
263   *                INCREMENT DEST   = no                                 *
264   *                DEST ADDRESS     = TFIFO SFR                          *
265   *                INCREMENT SOURCE = yes          (done in transmit routine) *
266   *                SOURCE ADDRESS   = xmt_buffer[] (done in transmit routine) *
267   *                                                                      *
268   *                The channels are not "active" until their GO bits are set. *
269   *                                                                      *
270   * Input:         gsc_dma_init();                                       *
271   *                                                                      *
272   * Output:        Nothing.                                              *
273   *                                                                      *
274   * Globals:       gcd_error    : module GCD.C                           *
275   *                80C152 regs : module REG152.H                         *
276   *                                                                      *
277   * Edit History: 07/10/90 - Written by Richard P. Smurlo.               *
278   *                                                                      *
279   \**********************************************************************/
280
281   static void gsc_dma_init()
282   {
283     gcd_error = AOK;                        /* Assume function successful...    */
284
285     /* DMA Initialization:                                                      */
286
287     DMA     = GSC_DMA;                      /* CPU or DMA control?              */
288     PCON    = PCON & HLDHLDA_DISABLE;       /* Disable the Hold/Hold Ack. Logic */
289
290     /* DMA Channel 0 Init. - Receiver:                                          */
291
292     DCON0   = 0x00;
```

```
293     DCON0   = DCON0 | R_XFERMODE_MASK;      /* Set Transfer Mode.              */
294     DCON0   = DCON0 | R_DMNDMODE_MASK;      /* Set Demand Mode.                */
295     DCON0   = DCON0 | R_SRC_AUTOINC_MASK;   /* Set up Auto Increment Option.   */
296     DCON0   = DCON0 | R_SOURCE_MASK;        /* Define DMA Source.              */
297     DCON0   = DCON0 | R_DST_AUTOINC_MASK;   /* Auto inc destination address.   */
298     DCON0   = DCON0 | R_DEST_MASK;          /* Define DMA Destination.         */
299
300     SARL0   = ADR_RFIFO;                    /* Source Address = RFIFO.         */
301     SARH0   = 0x00;
302
303     /* DMA Channel 1 Init. - Transmitter:                                      */
304
305     DCON1   = 0x00;
306     DCON1   = DCON1 | X_XFERMODE_MASK;      /* Transmit Transfer Mode.         */
307     DCON1   = DCON1 | X_DMNDMODE_MASK;      /* Set Demand Mode.                */
308     DCON1   = DCON1 | X_SRC_AUTOINC_MASK;   /* Set auto increment option.      */
309     DCON1   = DCON1 | X_SOURCE_MASK;        /* Define DMA Source.              */
310     DCON1   = DCON1 | X_DST_AUTOINC_MASK;   /* Auto inc source address.        */
311     DCON1   = DCON1 | X_DEST_MASK;          /* Define DMA Destination.         */
312
313     DARL1   = ADR_TFIFO;                    /* Dest. Addr. = TFIFO.            */
314     DARH1   = 0x00;
315
316     #if defined(DEBUG)
317     printf("gsc_dma_init: TSTAT = %02bx\n", TSTAT);
318     printf("gsc_dma_init: PCON  = %02bx\n", PCON);
319     printf("gsc_dma_init: DCON0 = %02bx\n", DCON0);
320     printf("gsc_dma_init: SARL0 = %02bx\n", SARL0);
321     printf("gsc_dma_init: SARH0 = %02bx\n", SARH0);
322     printf("gsc_dma_init: DCON1 = %02bx\n", DCON1);
323     printf("gsc_dma_init: DARL1 = %02bx\n", DARL1);
324     printf("gsc_dma_init: DARH1 = %02bx\n", DARH1);
325     #endif
326 }
327
328
329 /*************************************************************************\
330 *                          gsc_set_rcv_dst                                *
331 *************************************************************************
332 *                                                                         *
333 * Function:     Sets the DMA receiver destination address and byte count. *
334 *               DMA channel 0 is the receive channel. The destination for *
335 *               the reception is the parameter frame. Special care must be*
336 *               taken so that data in the destination is not over written *
337 *               (or processed) before the entire frame is received. The   *
338 *               destination reception addr MUST be of type XDATA.         *
339 *                                                                         *
340 * Input:        gsc_set_rcv_dst(                                          *
341 *                 gcd_frame f; frame to be received (must be type XDATA). *
342 *                 word length; frame length (0 < l <= GCD_MAX_FRAME_LENGTH).*
343 *               );                                                        *
344 *                                                                         *
345 * Output:       Nothing.                                                  *
346 *                                                                         *
347 * Globals:      gcd_error   : module GCD.C                                *
348 *               80C152 regs : module REG152.H                             *
349 *                                                                         *
350 * Edit History: 06/28/90 - Written by Robin T. Laird.                     *
351 *                                                                         *
352 \*************************************************************************/
353
354 static void gsc_set_rcv_dst(f, length)
355 gcd_frame f;
356 word length;
357 {
358   gcd_error = AOK;                      /* Assume function successful...   */
359
360   /* Set up destination and byte count registers.                          */
361
362   DARL0 = xptr_lo_offset(f);            /* Set DMA reception dest addr.    */
363   DARH0 = xptr_hi_offset(f);
364   BCRL0 = length & 0xFF;                /* Set frame length hi/lo byte cnt.*/
365   BCRH0 = (length >> 8) & 0xFF;
```



```
366
367   #if defined(DEBUG)
368   printf("gsc_set_rcv_dst: DAR(HL)0 = %bx %bx\n", DARH0, DARL0);
369   printf("gsc_set_rcv_dst: BCR(HL)0 = %bx %bx\n", BCRH0, BCRL0);
370   #endif
371 }
372
373
374 /*************************************************************************\
375 *                          gsc_set_xmt_src                                *
376 *************************************************************************
377 *                                                                         *
378 * Function:     Sets the DMA transmitter source address and byte count.   *
379 *               DMA channel 1 is the transmit channel. The source for the *
380 *               transmission is the parameter frame. Special care must be *
381 *               taken so that the source data is not over written before  *
382 *               the transmission is complete. The transmission source addr*
383 *               MUST be of type XDATA. The first byte of the frame holds  *
384 *               the transmission destination node address.                *
385 *                                                                         *
386 * Input:        gsc_set_xmt_src(                                          *
387 *                 gcd_frame f; frame to be received (must be type XDATA). *
388 *                 word length; frame length (0 < l <= GCD_MAX_FRAME_LENGTH).*
389 *               );                                                        *
390 *                                                                         *
391 * Output:       Nothing.                                                  *
392 *                                                                         *
393 * Globals:      gcd_error   : module GCD.C                                *
394 *               80C152 regs : module REG152.H                             *
395 *                                                                         *
396 * Edit History: 06/28/90 - Written by Robin T. Laird.                     *
397 *                                                                         *
398 \*************************************************************************/
399
400 static void gsc_set_xmt_src(f, length)
401 gcd_frame f;
402 word length;
403 {
404   gcd_error = AOK;                      /* Assume function successful...   */
405
406   /* Set up source and byte count registers.                               */
407
408   SARL1 = xptr_lo_offset(f);            /* Set DMA tranmission source addr.*/
409   SARH1 = xptr_hi_offset(f);
410   BCRL1 = length & 0xFF;                /* Set frame length hi/lo byte cnt.*/
411   BCRH1 = (length >> 8) & 0xFF;
412
413   #if defined(DEBUG)
414   printf("gsc_set_xmt_src: SAR(HL)1 = %bx %bx\n", SARH1, SARL1);
415   printf("gsc_set_xmt_src: BCR(HL)1 = %bx %bx\n", BCRH1, BCRL1);
416   #endif
417 }
418
419
420 /*************************************************************************\
421 *                          gsc_start_rcv                                  *
422 *************************************************************************
423 *                                                                         *
424 * Function:     Initiates reception of a data frame.                      *
425 *               DMA channel 0 is the receive channel. It is assumed that  *
426 *               the data receive destination address is already initialized.*
427 *               THIS FUNCTION MUST BE CALLED AFTER gsc_rcv_go(), as in:    *
428 *                                                                         *
429 *               gsc_rcv_go();     -- Set DMA GO bit.                      *
430 *               gsc_start_rcv(); -- Start receiver and enable interrupts. *
431 *                                                                         *
432 * Input:        gsc_start_rcv();                                          *
433 *                                                                         *
434 * Output:       Nothing.                                                  *
435 *                                                                         *
436 * Globals:      gcd_error   : module GCD.C                                *
437 *               80C152 regs : module REG152.H                             *
438 *                                                                         *
```

```
439   * Edit History: 07/10/90 - Written by Richard P. Smurlo and Robin T. Laird.  *
440   *                                                                             *
441   \*****************************************************************************/
442
443   #define MAX_RFIFO_READS     5             /* Read RFIFO 5 times to clear.     */
444
445   static void gsc_start_rcv()
446   {
447     char temp;
448     int  i;
449
450     gcd_error = AOK;                        /* Assume function successful...    */
451
452     /* Clear the receiver FIFO. Must read bytes until RFNE = 0.                   */
453
454     for (i = 0; RFNE != 0 && i < MAX_RFIFO_READS; i++) temp = RFIFO;
455     if (i >= MAX_RFIFO_READS)
456     {
457       gcd_error = ERR_RFIFO_NOT_CLEAR;
458       return;
459     }
460
461     /* Start reception...                                                        */
462
463     GREN  = 1;                              /* Enable the GSC receiver.         */
464     FGSRE = 1;                              /* Enable Receive Error ISR.        */
465     FGSRV = 1;                              /* Enable Receive Valid ISR.        */
466   }
467
468
469   /*****************************************************************************\
470   *                              gsc_start_xmt                                  *
471   *****************************************************************************
472   *                                                                             *
473   * Function:     Sets up transmission of data frames.                          *
474   *               DMA channel 1 is the transmit channel. It is assumed that     *
475   *               data transmit source address is already initialized.          *
476   *               THIS FUNCTION MUST BE CALLED BEFORE gsc_xmt_go(), as in:       *
477   *                                                                             *
478   *               gsc_start_xmt(); -- Start transmitter and enable interrupts.  *
479   *               gsc_xmt_go();    -- Set DMA GO bit.                           *
480   *                                                                             *
481   * Input:        gsc_start_xmt();                                              *
482   *                                                                             *
483   * Output:       Nothing.                                                      *
484   *                                                                             *
485   * Globals:      gcd_error    : module GCD.C                                   *
486   *               80C152 regs : module REG152.H                                 *
487   *                                                                             *
488   * Edit History: 07/10/90 - Written by Richard P. Smurlo and Robin T. Laird.   *
489   *                                                                             *
490   \*****************************************************************************/
491
492   static void gsc_start_xmt()
493   {
494     gcd_error = AOK;                        /* Assume function successful...    */
495
496     /* Start transmission...                                                     */
497
498     TEN   = 1;                              /* TEN = 1 to enable the xmitter.   */
499     EGSTE = 1;                              /* Enable Transmit Error ISR.       */
500     EGSTV = 1;                              /* Enable Transmit Valid ISR.       */
501   }
502
503
504   /*****************************************************************************\
505   *                               gsc_rcv_go                                    *
506   *****************************************************************************
507   *                                                                             *
508   * Function:     Sets the DMA GO bit.                                          *
509   *               DMA channel 0 is the receive channel. It is assumed that      *
510   *               the data receive destination address is already initialized.  *
511   *               THIS FUNCTION MUST BE CALLED BEFORE gsc_start_rcv().          *
```



```
512   *                                                                             *
513   * Input:        gsc_rcv_go();                                                 *
514   *                                                                             *
515   * Output:       Nothing.                                                      *
516   *                                                                             *
517   * Globals:      gcd_error    : module GCD.C                                   *
518   *               80C152 regs : module REG152.H                                 *
519   *                                                                             *
520   * Edit History: 07/10/90 - Written by Richard P. Smurlo and Robin T. Laird.   *
521   *                                                                             *
522   \*****************************************************************************/
523
524   static void gsc_rcv_go()
525   {
526     gcd_error = AOK;                        /* Assume function successful...    */
527
528     /* Start reception...                                                        */
529
530     DCON0 = DCON0 | 0x01;                   /* Set DMA GO bit.                  */
531   }
532
533
534   /*****************************************************************************\
535   *                               gsc_xmt_go                                    *
536   *****************************************************************************
537   *                                                                             *
538   * Function:     Starts transmission of data frames.                           *
539   *               DMA channel 1 is the transmit channel. It is assumed that     *
540   *               data transmit source address is already initialized.          *
541   *               THIS FUNCTION MUST BE CALLED AFTER gsc_start_xmt().           *
542   *                                                                             *
543   * Input:        gsc_xmt_go();                                                 *
544   *                                                                             *
545   * Output:       Nothing.                                                      *
546   *                                                                             *
547   * Globals:      gcd_error    : module GCD.C                                   *
548   *               80C152 regs : module REG152.H                                 *
549   *                                                                             *
550   * Edit History: 07/10/90 - Written by Richard P. Smurlo and Robin T. Laird.   *
551   *                                                                             *
552   \*****************************************************************************/
553
554   static void gsc_xmt_go()
555   {
556     gcd_error = AOK;                        /* Assume function successful...    */
557
558     /* Start transmission...                                                     */
559
560     DCON1 = DCON1 | 0x01;                   /* Set DMA GO bit.                  */
561   }
562
563
564   /*****************************************************************************\
565   *                              gsc_stop_rcv                                   *
566   *****************************************************************************
567   *                                                                             *
568   * Function:     Disables the GSC receiver by turning off the enable bit.      *
569   *               Both the receive error and receive valid interrupt routines   *
570   *               are disabled. Any incoming frame is dropped which will cause  *
571   *               a hardware acknowledge error at the other end.                *
572   *                                                                             *
573   * Input:        gsc_stop_rcv();                                               *
574   *                                                                             *
575   * Output:       Nothing.                                                      *
576   *                                                                             *
577   * Globals:      gcd_error    : module GCD.C                                   *
578   *               80C152 regs : module REG152.H                                 *
579   *                                                                             *
580   * Edit History: 07/10/90 - Written by Richard P. Smurlo.                      *
581   *                                                                             *
582   \*****************************************************************************/
583
584   static void gsc_stop_rcv()
```

```
585   {
586     gcd_error = AOK;                        /* Assume function successful...    */
587
588     GREN  = 0;                              /* GREN = 0 to disable the recvr.   */
589     EGSRE = 0;                              /* Disable Receive Error ISR.       */
590     EGSRV = 0;                              /* Disable Receive Valid ISR.       */
591     DCON0 = DCON0 & 0xFE;                   /* Re-set DMA GO bit.               */
592   }
593
594
595   /*************************************************************************\
596   *                              gsc_stop_xmt                               *
597   *************************************************************************
598   *                                                                       *
599   * Function:     Disables the GSC transmitter by turning off the enable bit.  *
600   *               Both the transmit error and transmit valid interrupt    *
601   *               routines are disabled. Any outgoing frame is terminated *
602   *               will cause a CRC or alignment reception error at the other  *
603   *               end.                                                    *
604   *                                                                       *
605   * Input:        gsc_stop_xmt();                                         *
606   *                                                                       *
607   * Output:       Nothing.                                                *
608   *                                                                       *
609   * Globals:      gcd_error    : module GCD.C                             *
610   *               80C152 regs  : module REG152.H                          *
611   *                                                                       *
612   * Edit History: 07/10/90 - Written by Richard P. Smurlo.                *
613   *                                                                       *
614   \*************************************************************************/
615
616   static void gsc_stop_xmt()
617   {
618     gcd_error = AOK;                        /* Assume function successful...    */
619
620     TEN   = 0;                              /* TEN = 0 to disable transmitter.  */
621     EGSTE = 0;                              /* Disable Transmit Error ISR.      */
622     EGSTV = 0;                              /* Disable Transmit Valid ISR.      */
623     DCON1 = DCON1 & 0xFE;                   /* Re-set DMA GO bit.               */
624   }
625
626
627   /*************************************************************************\
628   *                          gsc_enable_interrupts                          *
629   *************************************************************************
630   *                                                                       *
631   * Function:     Enables processor interrupts for the 8031/80152.        *
632   *                                                                       *
633   * Input:        gsc_enable_interrupts();                                *
634   *                                                                       *
635   * Output:       Nothing.                                                *
636   *                                                                       *
637   * Globals:      gcd_error    : module GCD.C                             *
638   *               80C152 regs  : module REG152.H                          *
639   *                                                                       *
640   * Edit History: 03/09/91 - Written by Robin T. Laird.                   *
641   *                                                                       *
642   \*************************************************************************/
643
644   static void gsc_enable_interrupts()
645   {
646     gcd_error = AOK;                        /* Assume function successful...    */
647     EA = 1;                                 /* Enable all interrupts.           */
648   }
649
650
651   /*************************************************************************\
652   *                              gsc_rcv_valid                              *
653   *************************************************************************
654   *                                                                       *
655   * Function:     Processes a valid receive interrupt (EGSRV) from the GSC.   *
656   *               This routine is called upon the completion of a valid frame *
657   *               reception (i.e., no OVERRUN, ABORT, ALIGNMENT, nor CRC      *
```



```
658   *               error). If HBA is being used, then an ACK will be sent in   *
659   *               response to receiving this data frame.                      *
660   *                                                                           *
661   *               This interrupt routine services the global rcv_buffer.      *
662   *               As valid frames are received, they are placed in the next   *
663   *               available position in the receive buffer. If the buffer is  *
664   *               full, then the last (current) frame is overwritten until    *
665   *               the buffer is no longer full and room is available for the  *
666   *               frame being received. The DMA channel receive destination   *
667   *               address is set to point to the next available location in   *
668   *               the receive buffer.                                         *
669   *                                                                           *
670   *               The valid frame received counter of the global module state *
671   *               structure is incremented. The GSC receiver is re-enabled    *
672   *               and the DMA receive channel GO bit is set for reception of  *
673   *               the next frame.                                             *
674   *                                                                           *
675   * Input:        gsc_rcv_valid() interrupt;                                  *
676   *                                                                           *
677   * Output:       Nothing.                                                    *
678   *                                                                           *
679   * Globals:      rcv_buffer   : module BMF.C                                 *
680   *               _gcd_state   : module GCD.C                                 *
681   *               80C152 regs  : module REG152.H                              *
682   *                                                                           *
683   * Edit History: 07/10/90 - Written by Richard P. Smurlo.                    *
684   *                                                                           *
685   \*************************************************************************/
686
687   static void gsc_rcv_valid() interrupt 5 using 2
688   {
689     /* Make sure buffer isn't already full.                                    */
690     /* If buffer full, incoming frames will be dropped, no ACK will be sent.   */
691
692     if (rcv_buffer.full)
693     {
694       GREN = 0;                             /* Turn receiver off - NO ACK.      */
695       return;
696     }
697
698     /* Determine length of received frame from DMA byte count registers.       */
699     /* The DMA receive byte count is initialized to GCD_MAX_FRAME_LENGTH.      */
700     /* As bytes are received, the byte count registers are decremented.        */
701
702     rcv_buffer.item[rcv_buffer.rear][GCD_LEN_POS] =
703       GCD_MAX_FRAME_LENGTH - ((BCRH0 << 8) | BCRL0);
704
705     /* Adjust rear index (MAX_BUFFER_SIZE-1 is last element in buffer).        */
706     /* Buffer can't be empty since we just received a frame.                   */
707
708     buf_inc(rcv_buffer.rear);
709
710     if (rcv_buffer.front == rcv_buffer.rear) rcv_buffer.full = TRUE;
711     rcv_buffer.empty = FALSE;
712
713     /* Increment valid reception count variable of module state structure.     */
714     /* Rolls over after 65535 frames received (if someone is keeping track).  */
715
716     _gcd_state.r_valid_cnt++;
717
718     /* Set receive destination to newly updated rear of buffer.                */
719     /* Reset DMA byte count registers for next reception.                       */
720
721     DARL0 = xptr_lo_offset(buf_rear(rcv_buffer));
722     DARH0 = xptr_hi_offset(buf_rear(rcv_buffer));
723     BCRL0 = GCD_MAX_FRAME_LENGTH & 0xFF;
724     BCRH0 = (GCD_MAX_FRAME_LENGTH >> 8) & 0xFF;
725
726     /* Start next reception...                                                 */
727
728     DCON0 = DCON0 | 0x01;                   /* Set DMA GO bit.                  */
729     GREN  = 1;                              /* Re-enable the GSC receiver.      */
730   }
```

```
731
732
733  /*************************************************************************\
734  *                              gsc_xmt_valid                              *
735  *************************************************************************
736  *                                                                         *
737  * Function:     Processes a valid transmit interrupt (EGSTV) from the GSC.*
738  *               This routine is called upon the completion of a valid frame*
739  *               transmission (i.e., no COLLISION DETECT, UNDERRUN, nor    *
740  *               ACKNOWLEDGE error). If HBA is being used, then an ACK would*
741  *               already have been received when we entered this routine.  *
742  *                                                                         *
743  *               This interrupt routine services the global xmt_buffer.    *
744  *               The current frame is removed from the transmit buffer. If *
745  *               the transmit buffer is empty, transmission interrupts and *
746  *               the transmitter are disabled. Otherwise, the transmitter is*
747  *               set up to transmit the next frame in the buffer. The DMA  *
748  *               channel transmit source address is set to point to the next*
749  *               location occupied in the transmit buffer.                 *
750  *                                                                         *
751  *               The valid frame transmit counter of the global module state*
752  *               structure is incremented. The GSC transmitter is re-enabled*
753  *               (so we can still respond to HBA) but the DMA transmit     *
754  *               channel GO bit is cleared since (currently) the transmit  *
755  *               buffer is conceptually only one deep.                     *
756  *                                                                         *
757  * Input:        gsc_xmt_valid() interrupt;                                *
758  *                                                                         *
759  * Output:       Nothing.                                                  *
760  *                                                                         *
761  * Globals:      xmt_buffer  : module BMF.C                                *
762  *               gcd_state   : module GCD.C                                *
763  *               80C152 regs : module REG152.H                             *
764  *                                                                         *
765  * Edit History: 07/10/90 - Written by Richard P. Smurlo.                  *
766  *                                                                         *
767  \*************************************************************************/
768
769  static void gsc_xmt_valid() interrupt 8 using 2
770  {
771    /* Buffer can't be full since we just removed an element.              */
772
773    xmt_buffer.full = FALSE;
774
775    /* Adjust front index (MAX_BUFFER_SIZE-1 is last element in buffer).   */
776
777    buf_inc(xmt_buffer.front);
778
779    /* Check and see if we've depleted the buffer (set empty flag if so).  */
780
781    if (xmt_buffer.front == xmt_buffer.rear) xmt_buffer.empty = TRUE;
782
783    /* Increment valid transmission count variable of module state structure. */
784    /* Rolls over after 65535 frames xmitted (if someone is keeping track).  */
785
786    _gcd_state.x_valid_cnt++;
787
788    /* If more than one element in transmit buffer...                      */
789    /*   Set transmit source to newly updated front of buffer.             */
790    /*   Reset DMA byte count registers for next transmission.             */
791    /*   Re-enable the transmitter (must be done BEFORE GO bit is set).    */
792    /*   Set DMA GO bit to start next transmission.                        */
793    /* Otherwise, we're done transmitting for now, so turn things off...   */
794    /*   Disable transmit valid and error interrupts.                      */
795    /*   Disable the DMA channel.                                          */
796    /*   But still enable the transmitter for HBA.                         */
797
798    if (!xmt_buffer.empty)
799    {
800      SARL1 = xptr_lo_offset(buf_front(xmt_buffer));
801      SARH1 = xptr_hi_offset(buf_front(xmt_buffer));
802      BCRL1 = xmt_buffer.item[xmt_buffer.front][GCD_LEN_POS] & 0xFF;
803      BCRH1 = (xmt_buffer.item[xmt_buffer.front][GCD_LEN_POS] >> 8) & 0xFF;
```



```
804      TEN   = 1;
805      DCON1 = DCON1 | 0x01;
806    }
807    else
808    {
809      EGSTV = 0;                    /* Disable transmit valid interrupts.  */
810      EGSTE = 0;                    /* Disable transmit error interrupts.  */
811      DCON1 = DCON1 & 0xFE;         /* Clear the DMA GO bit.               */
812      TEN   = 1;                    /* Re-enable the transmitter.          */
813    }
814  }
815
816
817  /*************************************************************************\
818  *                              gsc_rcv_error                              *
819  *************************************************************************
820  *                                                                         *
821  * Function:     Processes a receive error interrupt (EGSRE) from the GSC. *
822  *               The module state structure error variables are updated.   *
823  *               The DMA receive destination pointers and associated byte  *
824  *               count registers are re-initialized, and the GSC is set    *
825  *               to try and receive the frame again.                       *
826  *                                                                         *
827  * Input:        gsc_rcv_error() interrupt;                                *
828  *                                                                         *
829  * Output:       Nothing.                                                  *
830  *                                                                         *
831  * Globals:      _gcd_state  : module GCD.C                                *
832  *               80C152 regs : module REG152.H                             *
833  *                                                                         *
834  * Edit History: 07/10/90 - Written by Richard P. Smurlo.                  *
835  *                                                                         *
836  \*************************************************************************/
837  static void gsc_rcv_error() interrupt 6 using 2
838  {
839    int   i;
840    char  temp;
841
842    /* Log the type of receive error.                                      */
843    /* The sequence of checking is critical to correct error interpretation. */
844
845    _gcd_state.r_err_cnt++;
846    if (RSTAT & R_OVR_ERR_MASK)
847      _gcd_state.r_ovr_err_cnt++;
848    else
849      if (RSTAT & R_RCABT_ERR_MASK)
850        _gcd_state.r_rcabt_err_cnt++;
851      else
852        if (RSTAT & R_AE_ERR_MASK)
853          _gcd_state.r_ae_err_cnt++;
854        else
855          if (RSTAT & R_CRCE_ERR_MASK)
856            _gcd_state.r_crce_err_cnt++;
857
858    /* Just had a bad reception, so set things up to receive again.        */
859    /* Must clear the receiver FIFO (read bytes until RFNE = 0).           */
860    /* Must re-initialize the receive destination DMA pointers and byte count.*/
861    /* Incoming frames still go to the end of the receive buffer.          */
862
863    for (i = 0; RFNE != 0 && i <= MAX_RFIFO_READS; i++) temp = RFIFO;
864
865    DARL0 = xptr_lo_offset(buf_rear(rcv_buffer));
866    DARH0 = xptr_hi_offset(buf_rear(rcv_buffer));
867    BCRL0 = GCD_MAX_FRAME_LENGTH & 0xFF;
868    BCRH0 = (GCD_MAX_FRAME_LENGTH >> 8) & 0xFF;
869
870    DCON0 = DCON0 | 0x01;               /* Set DMA Go bit.                 */
871    GREN  = 1;                          /* Re-enable the GSC receiver.     */
872  }
873
874
875  /*************************************************************************\
876  *                              gsc_xmt_error                              *
```

```
877  *****************************************************************************
878  *                                                                           *
879  * Function:     Processes a transmit error interrupt (EGSTE) from the GSC.  *
880  *               The module state structure error variables are updated.     *
881  *               The DMA transmit source pointers and associated byte count  *
882  *               registers are re-initialized, and the GSC is set to try     *
883  *               and transmit the frame again.                               *
884  *                                                                           *
885  * Input:        gsc_xmt_error() interrupt;                                  *
886  *                                                                           *
887  * Output:       Nothing.                                                    *
888  *                                                                           *
889  * Globals:      _gcd_state  : module GCD.C                                  *
890  *               gcd_tries   : module GCD.C                                  *
891  *               gcd_times   : module GCD.C                                  *
892  *               gcd_stop    : module GCD.C                                  *
893  *               80C152 regs : module REG152.H                               *
894  *                                                                           *
895  * Edit History: 07/10/90 - Written by Richard P. Smurlo.                    *
896  *               01/30/91 - Robin T. Laird added re-transmit count/stop.     *
897  *                                                                           *
898  \*****************************************************************************/
899
900  static void gsc_xmt_error() interrupt 9 using 2
901  {
902    /* Log the type of transmit error.                                    */
903    /* The sequence of checking is critical to correct error interpretation.  */
904
905     gcd_state.x_err_cnt++;
906    if (TSTAT & X_TCDT_ERR_MASK)
907      _gcd_state.x_tcdt_err_cnt++;
908    else
909      if (TSTAT & X_UR_ERR_MASK)
910        _gcd_state.x_ur_err_cnt++;
911      else
912        if (TSTAT & X_NOACK_ERR_MASK)
913          _gcd_state.x_noack_err_cnt++;
914
915    /* Check number of attempts made so far. If gcd_times == 0 not counting.  */
916    /* Variable gcd_times == 0 if attempt re-transmission forever.            */
917
918    if (gcd_times)
919    {
920      if (gcd_stop)
921      {
922        return;
923      }
924      else if (++gcd_tries >= gcd_times)
925      {
926        gcd_stop = TRUE;                  /* Hit re-transmit limit.          */
927        BCRH1 = 0;                        /* Set high byte count to 0.       */
928        BCRL1 = 0;                        /* Set low  byte count to 0.       */
929        TEN   = 1;                        /* Re-enable transmitter for ACKs. */
930        return;
931      }
932    }
933
934    /* Just had a bad transmission, so set things up to transmit again.       */
935    /* Must re-initialize the transmit source DMA pointers and byte count.    */
936    /* Outgoing frame is still at the front of the transmit buffer.           */
937
938    SARL1 = xptr_lo_offset(buf_front(xmt_buffer));
939    SARH1 = xptr_hi_offset(buf_front(xmt_buffer));
940    BCRL1 = xmt_buffer.item[xmt_buffer.front][GCD_LEN_POS] & 0xFF;
941    BCRH1 = (xmt_buffer.item[xmt_buffer.front][GCD_LEN_POS] >> 8) & 0xFF;
942
943    TEN   = 1;                            /* TEN = 1 to enable the xmitter.  */
944    while(!TEN);                          /* Wait for TEN to actually be set.*/
945    DCON1 = DCON1 | 0x01;                 /* Set DMA GO bit.                 */
946  }
```

```
#*****************************************************************************#
#                                  MAKEFILE                                   #
#*****************************************************************************#
#                                                                             #
# CPCI:          IED90-MRA-MPU-AC-MAKFFILE-TXT-R0C0                           #
#                                                                             #
# Description:   Makefile for the Modular Robotic Architecture (MRA).         #
#                Makes the MPU applications controller subsystem.             #
#                                                                             #
#                Targets are available for the following systems/subsystems:  #
#                                                                             #
#                mac    - MPU Applications Controller                         #
#                print  - Print MPU AC source files                           #
#                                                                             #
# Notes:         1) The dependency and production rules are included here.     #
#                2) See also \mra\makefile.                                   #
#                                                                             #
# Edit History: 03/25/91 - Written by Robin T. Laird.                         #
#                                                                             #
#*****************************************************************************#


########################################## RULES ##########################################

.SUFFIXES : .hex .exe .obj .c .a51

# Control settings for Franklin 8031 development

CC          =c51
AS          =a51
LINK        =l51
OTOH        =oh851
CFLAGS      =cd la db sb
ASFLAGS     =
LFLAGS      =
OFLAGS      =
STARTUP     =\c51\crom.obj
CODESEG     =00000h
XDATASEG    =00000h

# Control settings for Microsoft MS-DOS development

MSC         =cl
MSAS        =masm
MSLINK      =link
MSCFLAGS    =/AL /c /O1 /Z1 /Od
MSASFLAGS   =
MSLNKFLGS   =/co
LOADLIBES   =

.c.obj :
        $(CC) $< $(CFLAGS)

.a51.obj :
        $(AS) $< $(ASFLAGS)

.obj.exe :
        $(LINK) $(STARTUP),$< TO $@ code ($(CODESEG)) xdata ($(XDATASEG)) ixref

.exe.hex :
        $(OTOH) $< $(OFLAGS)



########################################## DEFINITIONS ##########################################

# Project, system, and application level definitions

PROJ            = mra
APPSYS          = app
COMSYS          = com
ICNSYS          = icn
```



```
MPUSYS          = mpu

APPSRC          = \$(PROJ)\$(APPSYS)\src
COMSRC          = \$(PROJ)\$(COMSYS)\src
ICNSRC          = \$(PROJ)\$(ICNSYS)\src
MPUSRC          = \$(PROJ)\$(MPUSYS)\src

COMBINMS        = \$(PROJ)\$(COMSYS)\bin\msdos

MPUBIN31        = \$(PROJ)\$(MPUSYS)\bin\8031
MPUBIN152       = \$(PROJ)\$(MPUSYS)\bin\80152
MPUBINMS        = \$(PROJ)\$(MPUSYS)\bin\msdos
MPUBINSBC8      = \$(PROJ)\$(MPUSYS)\bin\sbc8

# Common subsystem level source directories

DEVSRC          = $(COMSRC)\dev
HDRSRC          = $(COMSRC)\hdr
LCSSRC          = $(COMSRC)\lcs
MMSSRC          = $(COMSRC)\mms

# ICN subsystem level source directories

GCSSRC          = $(ICNSRC)\gcs
IACSRC          = $(ICNSRC)\ac

# MPU subsystem level source directories

LDSSRC          = $(MPUSRC)\lds
MACSRC          = $(MPUSRC)\ac

# Common subsystem global include and compilation units

SYSDEFS         = $(HDRSRC)\sysdefs.h

# MPU subsystem compilation units

MAC             = $(MPUBIN31)\main.obj      $(MPUBIN31)\mra.obj       \
                  $(MPUBINMS)\main.obj      $(MPUBINMS)\mra.obj



########################################## TARGETS ##########################################

mac   : $(MAC)

print : $(MAC)
        %-a2ps -nf main.c | post
        %-a2ps -nf mra.h | post
        %-a2ps -nf mra.c | post
        touch print



############################## MPU MAC DEPENDENCIES ##############################

# MPU MAC main program dependencies

MPUMAIN = $(SYSDEFS)            $(MACSRC)\mra.h         $(GCSSRC)\gci.h \
          $(GCSSRC)\gcd.h       $(LCSSRC)\lci.h         $(LCSSRC)\lcd.h \
          $(MMSSRC)\mm.h        $(MMSSRC)\lmm.h         $(MMSSRC)\smm.h \
          $(LDSSRC)\ldi.h       $(DEVSRC)\rtc.h         $(MACSRC)\main.c

$(MPUBIN31)\main.obj            : $(MPUMAIN)
  $(CC) $(MACSRC)\$*.c $(CFLAGS) df(I8031) pr($(MACSRC)\$*.31) oj($(MPUBIN31)\$

$(MPUBINMS)\main.obj            : $(MPUMAIN)
  $(MSC) $(MSCFLAGS) /DIBMAT /Fs$(MACSRC)\$*.at /Fo$(MPUBINMS)\$* $(MACSRC)\$*.

# MPU MAC mra system dependencies

MPUMRA = $(SYSDEFS)             $(MACSRC)\mra.h         $(GCSSRC)\gci.h \
```

```
147             $(GCSSRC)\gcd.h         $(LCSSRC)\lci.h         $(LCSSRC)\lcd.h \
148             $(MMSSRC)\mm.h          $(MMSSRC)\lmm.h         $(MMSSRC)\smm.h \
149             $(LDSSRC)\ldl.h         $(MACSRC)\mra.c
150
151   $(MPUBIN31)\mra.obj               : $(MPUMRA)
152     $(CC) $(MACSRC)\$*.c $(CFLAGS) df(18031) pr($(MACSRC)\$*.31) oj($(MPUBIN31)\$
153
154   $(MPUBINMS)\mra.obj               : $(MPUMRA)
155     $(MSC) $(MSCFLAGS) /DIBMAT /Fa$(MACSRC)\$*.at /Fo$(MPUBINMS)\$* $(MACSRC)\$*.
```

```
/*****************************************************************************\
*                                MAIN.C                                        *
*******************************************************************************
*                                                                             *
* CPCI:         IED90-MRA-MPU-AC-MAIN-C-R1C1                                  *
*                                                                             *
* Description:  MPU main program.                                             *
*               Implements the MRA MPU (user) application program.            *
*               This is the main program for the MPU system.                  *
*               It simply calls the standard MRA function mra_init() which    *
*               is responsible for initializing and coordinating the MRA      *
*               subsystems for the MPU application.                           *
*                                                                             *
* Notes:        1, Subsystems are selected by setting the associated USE_ss   *
*                  variable to YES (1). This can be done from either the      *
*                  compilation command line or by modifying the MRA.H file.   *
*                                                                             *
* Edit History: 10/10/90 - Written by Robin T. Laird.                         *
*                                                                             *
\*****************************************************************************/

#include <sysdefs.h>                  /* MRA standard declarations.      */
#include <rtc.h>                      /* MRA real-time clock functions.  */
#include "mra.h"                      /* MRA public literals/functions.  */

#include <debug.h>

#define TWO_SECONDS     2000L         /* Wait two seconds before start.  */

void main()
{
  /* Wait for a few seconds before fi     ip.                            */
  /* This gives the ICN time to init      ubsystems, buffers, etc.       */

  rtc_init();
  rtc_wait(TWO_SECONDS);

  /* Initialize the MRA subsystems and call mra_main().                  */
  /* Control never returns for systems with USE_MRA set to YES.          */

  mra_init();
}
```

```
/*****************************************************************************\
*                                   MRA.H                                     *
*****************************************************************************
*                                                                           *
* CPCI:          IED90-MRA-MPU-AC-MRA-H-R0C0                                *
*                                                                           *
* Description:   System configuration and default controller definitions.   *
*                Contains external declarations for the system initialization *
*                function and default application controller, mra_main().   *
*                                                                           *
*                The user/developer should select/de-select those MRA       *
*                subsystems that are required or being used. Only those     *
*                subsystems that are selected are initialized by mra_init().*
*                                                                           *
*                Selecting USE_MRA will cause the default system application *
*                controller, mra_main(), to be used. The init routine calls *
*                the default controller after all selected subsystems have  *
*                been initialized. If selected, mra_main() takes control and *
*                does not return.                                           *
*                                                                           *
*                Module MRA exports the following variables/functions:      *
*                                                                           *
*                int mra_error;                                             *
*                                                                           *
*                mra_init();                                                *
*                mra_main();                                                *
*                                                                           *
* Notes:         1) This file should be included only by the main() program. *
*                                                                           *
* Edit History: 07/07/90 - Written by Robin T. Laird.                       *
*                                                                           *
\*****************************************************************************/

#ifndef MRA_MODULE_CODE
#define MRA_MODULE_CODE         13000

/* Public Data Structures:                                                  */

#define ERR_MRA_NOT_INIT        1+MRA_MODULE_CODE

#define USE_GCS                 NO      /* Global Communications Subsystem. */
#define USE_LCS                 YES     /* Local  Communications Subsystem. */
#define USE_MMS                 YES     /* Method Management Subsystem.     */
#define USE_LDS                 NO      /* Logical Device Subsystem.        */
#define USE_MRA                 YES     /* Modular Robotic Architecture AC. */

#if USE_GCS
#include <gci.h>                        /* Global Comms Interface.          */
#include <gcd.h>                        /* Global Comms Device Driver.      */
#endif

#if USE_LCS
#include <lci.h>                        /* Local Comms Interface.           */
#include <lcd.h>                        /* Local Comms Device Driver.       */
#endif

#if USE_MMS
#include <mm.h>                         /* Method Manager.                  */
#include <pb.h>                         /* Phone Book.                      */
#include <lmm.h>                        /* Local Method Manager.            */
#include <smm.h>                        /* System Method Manager.           */
#endif

#if USE_LDS
#include <ldi.h>                        /* Logical Device Interface.        */
#endif

/* External module global error variable.                                   */

extern int mra_error;

/* Public Functions:                                                        */

```



```
void mra_init(void);
void mra_main(void);

#endif
```

```
1   /*****************************************************************************\
2   *                                   MRA.C                                     *
3   *****************************************************************************
4   *                                                                             *
5   * CPCI:          IED90-MRA-MPU-AC-MRA-C-R0C0                                  *
6   *                                                                             *
7   * Description:   MRA system initialization and default controller functions.  *
8   *                Implements the MRA system initialization and default system  *
9   *                application controller (AC) functions which represent the    *
10  *                highest level interface to the Modular Robotic Architecture  *
11  *                software systems. The mra_init() function must be called to  *
12  *                correctly initialize the various software subsystems. The    *
13  *                function mra_main() is the default AC and replaces the user  *
14  *                application program (mra_main() never returns to the calling *
15  *                function).                                                   *
16  *                                                                             *
17  *                Module MRA exports the following variables/functions:        *
18  *                                                                             *
19  *                int mra_error;                                               *
20  *                                                                             *
21  *                mra_init();                                                  *
22  *                mra_main();                                                  *
23  *                                                                             *
24  * Notes:         1) The MRA functions are implementation independent.         *
25  *                2) Module MRA represents the Default Applications Controller.*
26  *                                                                             *
27  * Edit History: 02/04/91 - Written by Robin T. Laird.                         *
28  *                                                                             *
29  \*****************************************************************************/
30
31  #include <sysdefs.h>                    /* System constants and types.      */
32  #include "mra.h"                        /* MRA public literals/functions.   */
33
34  /* Public Variables:                                                        */
35
36  /* Global module error variable, mra_error.                                */
37  /* mra_error contains code of last error occurence.                        */
38  /* Should be set to AOK after each successful function call.               */
39  /* Variable can be examined by other software after each function call.    */
40
41  XDATA int mra_error = ERR_MRA_NOT_INIT;
42
43
44  /*****************************************************************************\
45  *                                  mra_init                                   *
46  *****************************************************************************
47  *                                                                             *
48  * Function:      Initializes the MRA software subsystems.                     *
49  *                The subsystems are selected in the file MRA.H and only those *
50  *                subsystems selected will be initialized. If the default      *
51  *                application controller mra_main() is selected then a call is *
52  *                made to that function and control never returns. The default *
53  *                AC can be called separately by a call to mra_main() after    *
54  *                mra_init() returns (the same result is acheived).            *
55  *                                                                             *
56  * Input:         mra_init();                                                  *
57  *                                                                             *
58  * Output:        Nothing.                                                     *
59  *                                                                             *
60  * Globals:       mra_error : module MRA.C                                     *
61  *                gci_error : module GCI.C                                     *
62  *                lci_error : module LCI.C                                     *
63  *                mm_error  : module MM.C                                      *
64  *                ldi_error : module LDI.C                                     *
65  *                                                                             *
66  * Edit History: 10/01/90 - Written by Robin T. Laird.                         *
67  *                                                                             *
68  \*****************************************************************************/
69
70  void mra_init()
71  {
72    /* Initialize selected subsystems, return upon detected failure.          */
73
```



```
74    #if USE_GCS
75    gci_init();
76    if (gci_error != AOK) return;
77    #endif
78
79    #if USE_LCS
80    lci_init();
81    if (lci_error != AOK) return;
82    #endif
83
84    #if USE_MMS
85    mm_init();
86    if (mm_error != AOK) return;
87    #endif
88
89    #if USE_LDS
90    ldi_init();
91    if (ldi_error != AOK) return;
92    #endif
93
94    /* Function successful...                                                 */
95    /* Set error variable to OK and return only if USE_MRA not selected.      */
96
97    mra_error = AOK;
98
99    /* Call MRA main program and never return...                              */
100
101   #if USE_MRA
102   mra_main();
103   #endif
104 }
105
106
107 /*****************************************************************************\
108 *                                  mra_main                                   *
109 *****************************************************************************
110 *                                                                             *
111 * Function:      Default Application Controller (AC) for the MPU system.      *
112 *                The main() C program calls mra_init() and which then calls   *
113 *                mra_main(). The default controller coordinates operation of  *
114 *                the MRA subsystems to pass information from the LAN to the   *
115 *                module processor (MPU) and vice versa.                       *
116 *                                                                             *
117 * Input:         mra_main();                                                  *
118 *                                                                             *
119 * Output:        Nothing.                                                     *
120 *                                                                             *
121 * Globals:       None.                                                        *
122 *                                                                             *
123 * Edit History: 02/04/91 - Written by Robin T. Laird.                         *
124 *                                                                             *
125 \*****************************************************************************/
126
127 void mra_main()
128 {
129   /* Cycle the method manager forever.                                      */
130   /* Messages will be received via the LCI and processed according to type. */
131   /* If dictionary functions activate system methods then they must call    */
132   /* mm_cycle() "occasionally" so that incoming messages are not dropped.   */
133
134   #if USE_MRA
135   mm_cycle(MM_CYCLE_FOREVER);
136   #endif
137 }
```

```
#*****************************************************************************#
#                                MAKEFILE                                     #
#*****************************************************************************#
#                                                                             #
# CPCI:          IED90-MRA-MPU-LDS-MAKEFILE-TXT-R0C0                          #
#                                                                             #
# Description:   Makefile for the Modular Robotic Architecture (MRA).         #
#                Makes the logical device interface subsystem.                #
#                                                                             #
#                Targets are available for the following systems/subsystems:  #
#                                                                             #
#                lds    - Logical Device Interface Subsystem                  #
#                lib    - Add modules to MRA library                          #
#                print  - Add modules to MRA library                          #
#                                                                             #
# Notes:         1) The dependency and production rules are included here.     #
#                2) See also \mra\makefile.                                   #
#                                                                             #
# Edit History: 05/30/91 - Written by Robin T. Laird.                         #
#                                                                             #
#*****************************************************************************#


########################################## RULES ##########################################

.SUFFIXES : .hex .exe .obj .c .a51

# Control settings for Franklin 8031 development

CC          =c51
AS          =a51
LINK        =l51
OTOH        =ohs51
CFLAGS      =cd la db sb
ASFLAGS     =
LFLAGS      =
OFLAGS      =
STARTUP     =\c51\crom.obj
CODESEG     =00000h
XDATASEG    =00000h

# Control settings for Microsoft MS-DOS development

MSC         =cl
MSAS        =masm
MSLINK      =link
MSCFLAGS    =/AS /c /Oi /Zi /Od
MSASFLAGS   =
MSLNKFLGS   =/co
LOADLIBES   =

.c.obj :
        $(CC) $< $(CFLAGS)

.a51.obj :
        $(AS) $< $(ASFLAGS)

.obj.exe :
        $(LINK) $(STARTUP),$< TO $@ code ($(CODESEG)) xdata ($(XDATASEG)) ixref

.exe.hex :
        $(OTOH) $< $(OFLAGS)


########################################## DEFINITIONS ##########################################

# Project, system, and application level definitions

PROJ            = mra
APPSYS          = app
COMSYS          = com
```



```
MPUSYS          = mpu

MPULIB          = \$(PROJ)\lib
COMSRC          = \$(PROJ)\$(COMSYS)\src
MPUSRC          = \$(PROJ)\$(MPUSYS)\src

MPUBIN31        = \$(PROJ)\$(MPUSYS)\bin\8031
MPUBIN152       = \$(PROJ)\$(MPUSYS)\bin\80152
MPUBINMS        = \$(PROJ)\$(MPUSYS)\bin\msdos
MPUBINSBC8      = \$(PROJ)\$(MPUSYS)\bin\sbc8

# Common subsystem level source directories

HDRSRC          = $(COMSRC)\hdr

# Logical device interface subsystem level source directories

LDSSRC          = $(MPUSRC)\lds

# Common subsystem global include and compilation units

SYSDEFS         = $(HDRSRC)\sysdefs.h

# MPU subsystem compilation units

LDS             = $(MPUBIN152)\ldi.obj          $(MPUBIN31)\ldi.obj     \
                  $(MPUBINMS)\ldi.obj           $(MPUBINSBC8)\ldi.obj


########################################## TARGETS ##########################################

lds             : $(LDS)

lib             : $(LDS)
                  -lib51 delete $(MPULIB)\mra_1521.lib (ldi)
                  -lib51 add $(MPUBIN152)\ldi.obj to $(MPULIB)\mra_1521.lib
                  -lib51 delete $(MPULIB)\mra_311.lib (ldi)
                  -lib51 add $(MPUBIN31)\ldi.obj to $(MPULIB)\mra_311.lib
                  -lib $(MPULIB)\mra_mss.lib -ldi.obj+$(MPUBINMS)\ldi.obj;
                  -lib $(MPULIB)\mra_sbc8.lib -ldi.obj+$(MPUBINSBC8)\ldi.obj;
                  touch lib

print           : $(LDS)
                  %-a2ps -nf ldi.h | post
                  %-a2ps -nf ldi.c | post
                  touch print


############################## MPU DEV DEPENDENCIES ##############################

# MPU Logical Device Interface module dependencies for 80152, 8031, MS-DOS, SBC

LDI = $(SYSDEFS) $(LDSSRC)\ldi.h $(LDSSRC)\ldi.c

$(MPUBIN152)\ldi.obj            : $(LDI)
  $(CC) $(LDSSRC)\$*.c $(CFLAGS) df(I80152) pr($(LDSSRC)\$*.152) oj($(MPUBIN152

$(MPUBIN31)\ldi.obj             : $(LDI)
  $(CC) $(LDSSRC)\$*.c $(CFLAGS) df(I8031)  pr($(LDSSRC)\$*.31)  oj($(MPUBIN31)

$(MPUBINMS)\ldi.obj             : $(LDI)
  $(MSC) $(MSCFLAGS) /DIBMAT /Fs$(LDSSRC)\$*.at /Fo$(MPUBINMS)\$* $(LDSSRC)\$*.

$(MPUBINSBC8)\ldi.obj           : $(LDI)
  $(MSC) $(MSCFLAGS) /DSBC8 /Fs$(LDSSRC)\$*.sbc /Fo$(MPUBINSBC8)\$* $(LDSSRC)\$
```

```
1     /*****************************************************************************\
2     *                                  LDI.H                                      *
3     *****************************************************************************
4     *                                                                            *
5     * CPCI:          IED90-MRA-MPU-LDS-LDI-H-R0C0                                *
6     *                                                                            *
7     * Description:   Logical Device Interface (LDI) variables and functions.     *
8     *                Contains constant function parameter declarations (#defines) *
9     *                as well as function return values (for success and failure  *
10    *                of all operations). Contains the function prototypes for the *
11    *                LDI.C module.                                               *
12    *                                                                            *
13    *                Module LDI exports the following types/variables/functions: *
14    *                                                                            *
15    *                int ldi_error;                                              *
16    *                                                                            *
17    *                ldi_init();                                                 *
18    *                                                                            *
19    * Notes:         1) See SDS pp. 5-6 through 5-x for more information.        *
20    *                                                                            *
21    * Edit History: 03/25/91 - Written by Robin T. Laird.                        *
22    *                                                                            *
23    \*****************************************************************************/
24
25    #ifndef LDI_MODULE_CODE
26    #define LDI_MODULE_CODE                 14000
27
28    /* Public Data Structures:                                                  */
29
30    #define LDI_ERR_NOT_INIT                1+LDI_MODULE_CODE
31
32    /* External module global error variable.                                  */
33
34    extern int ldi_error;
35
36    /* Public Functions:                                                        */
37
38    void ldi_init();
39
40    #endif
```

```
1   /*****************************************************************************\
2   *                                  LDI.C                                      *
3   *****************************************************************************
4   *                                                                            *
5   * CPCI:          IED90-MRA-MPU-LDS-LDI-C-R0C0                                *
6   *                                                                            *
7   * Description:   Logical device interface (LDI) functions.                   *
8   *                Implements the standard MRA logical device interface module.*
9   *                The LDI provides functions for creating, deleting, and      *
10  *                manipulating abstract data types that represent logical     *
11  *                devices such as logical actuators or sensors.               *
12  *                                                                            *
13  *                Module LDI exports the following types/variables/functions: *
14  *                                                                            *
15  *                int ldi_error;                                              *
16  *                                                                            *
17  *                ldi_init();                                                 *
18  *                                                                            *
19  * Notes:         1) The LDI is implemented as a blackboard data structure.   *
20  *                                                                            *
21  * Edit History: 03/25/91 - Written by Robin T. Laird.                        *
22  *                                                                            *
23  \*****************************************************************************/
24
25  #include <sysdefs.h>                    /* System constants and types.      */
26  #include "ldi.h"                        /* Logical device interface.        */
27
28  /* Public Variables:                                                        */
29
30  /* Global module error variable, ldi_error.                                 */
31  /* ldi_error contains code of last error occurence.                         */
32  /* Should be set to AOK after each successful function call.                */
33  /* Variable can be examined by other software after each function call.     */
34
35  XDATA int ldi_error = LDI_ERR_NOT_INIT; /* Global module error variable.    */
36
37
38  /*****************************************************************************\
39  *                                ldi_init                                    *
40  *****************************************************************************
41  *                                                                            *
42  * Function:      Initializes the Logical Device Interface software subsystems.*
43  *                This includes initializing data structures such as the      *
44  *                system blackboard and the memory allocation routines.       *
45  *                                                                            *
46  * Input:         ldi_init();                                                 *
47  *                                                                            *
48  * Output:        Nothing.                                                    *
49  *                                                                            *
50  * Globals:       ldi_error : LDI.C                                           *
51  *                                                                            *
52  * Edit History: 03/25/91 - Written by Robin T. Laird.                        *
53  *                                                                            *
54  \*****************************************************************************/
55
56  void ldi_init()
57  {
58    ldi_error = AOK;                      /* Assume function successful.      */
59  }
60
```

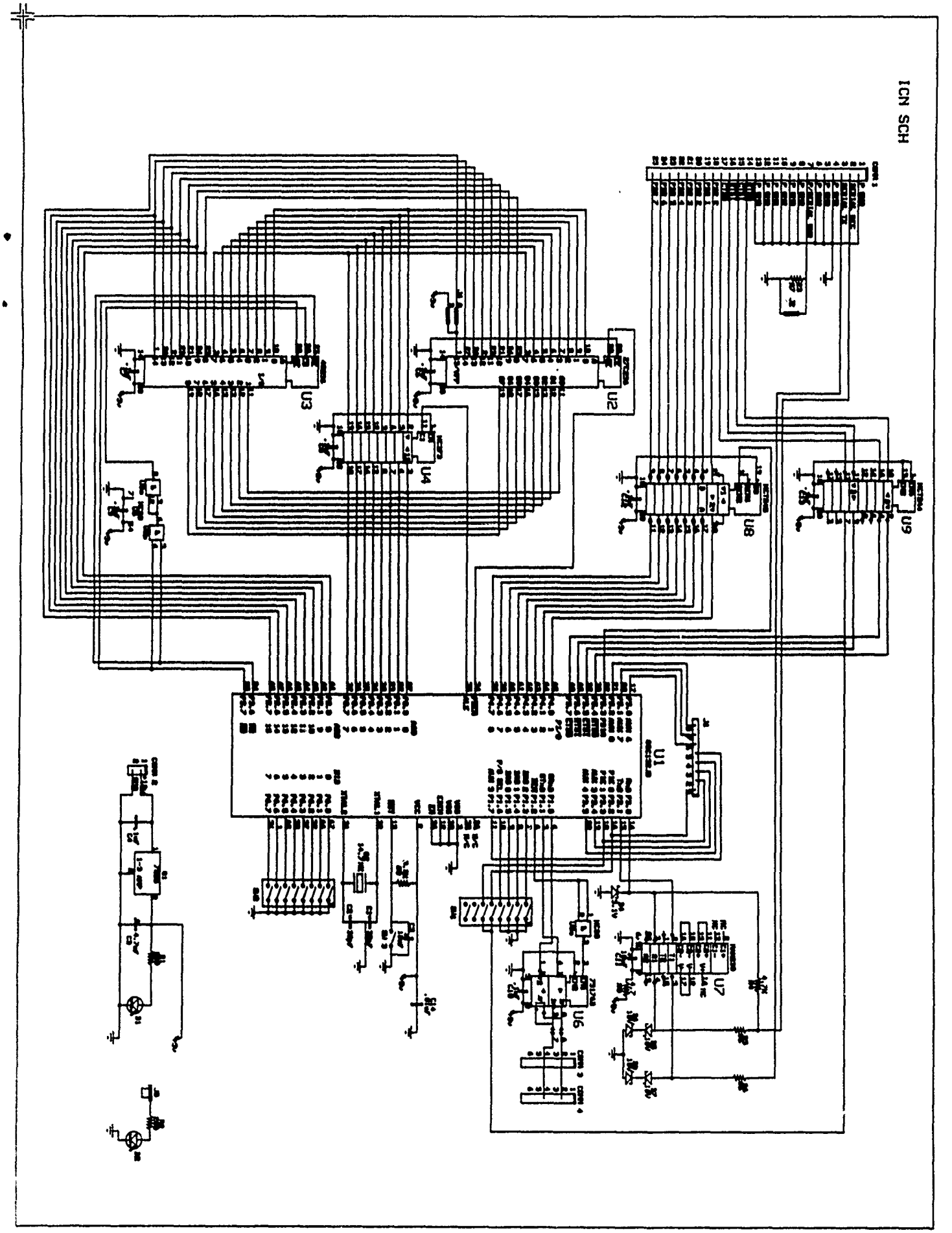
APPENDIX B

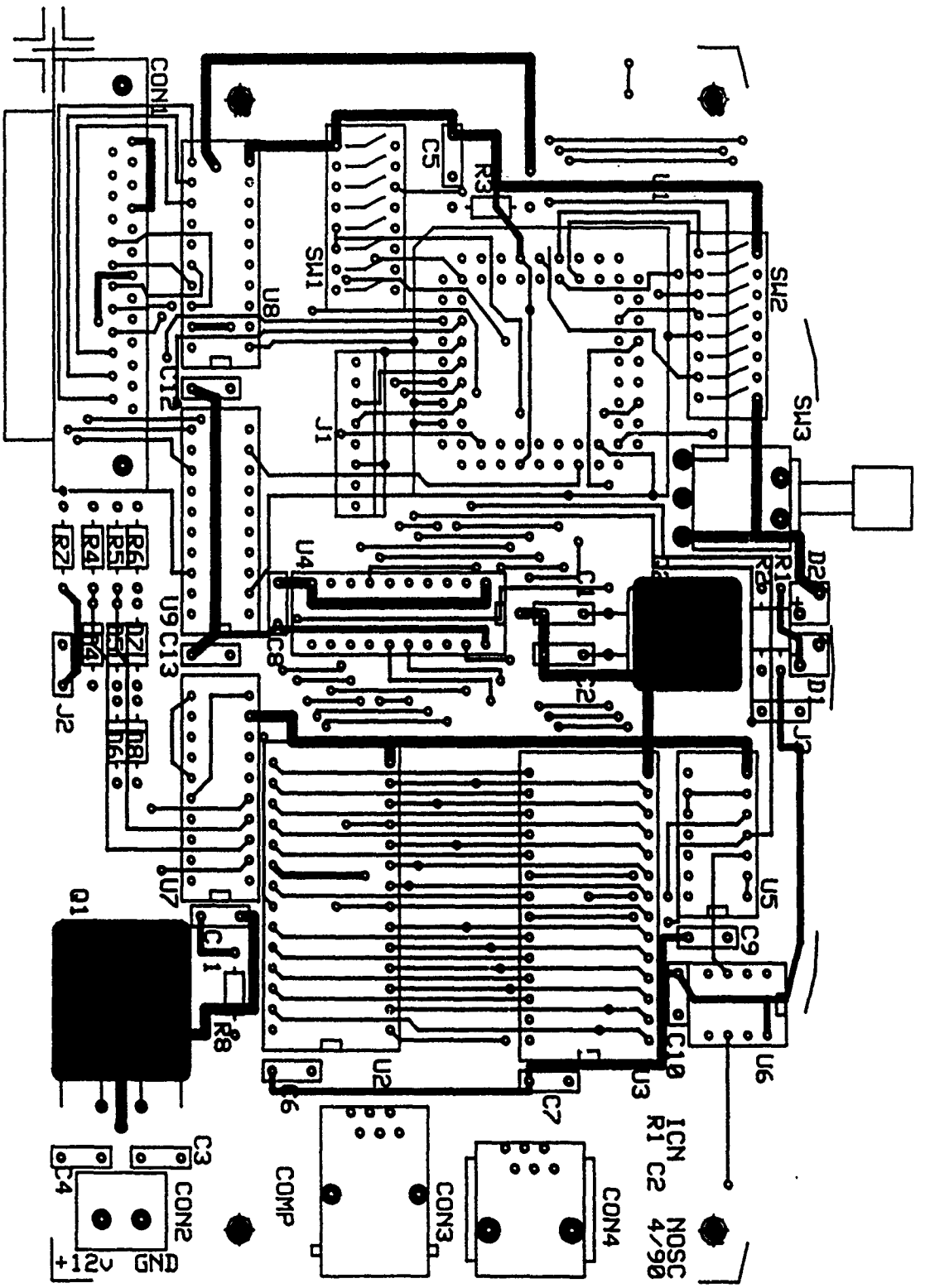
HARDWARE SYSTEM IMPLEMENTATION

Appendix B. Hardware System Implementation

The following section provides details on the standard hardware subsystems of the MRA. The information provided herein is sufficient to implement the standard hardware components that each robot module possesses.

For each of the standard components (i.e., ICN, PDN, and the PPCU), a schematic, layout diagram, and a parts list are included.


ICN SCH
U1
U2
U3
U4
U6
U7
U8
U9


CON1
C5
R3
U1
SW1
U8
SW2
SW3
C12
J1
R2
R5
R6
U4
R1
D2
D1
U9
C13
C8
C2
J2
U7
U5
C9
Q1
C1
R8
U2
C6
C7
U3
C10
U6
C3
C4
CON2
+12v
GND
COMP
CON3
CON4
ICN
R1 C2
NOSC
4/90

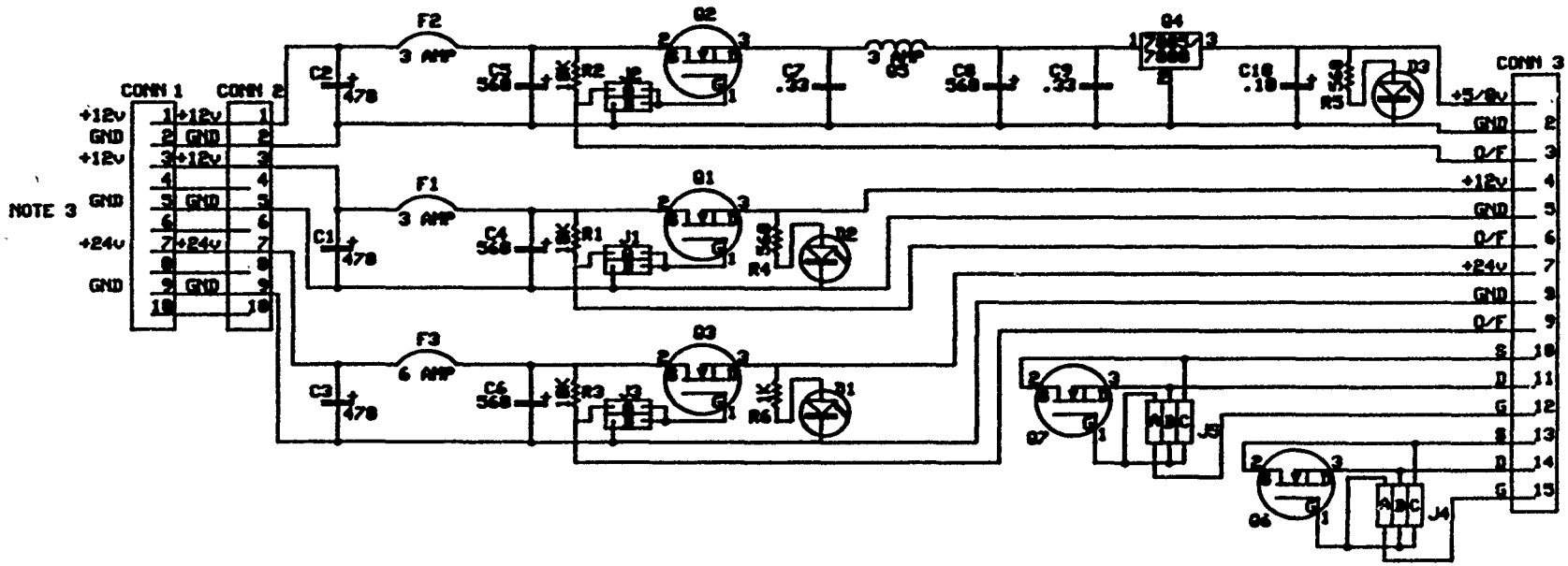
Intelligent Communications Node (ICN) Parts List

Board Ref.	Quan.	P/N - Discription
U1	1	N80C152JB-1, MPU
U2	1	27C256, EPROM
U3	1	43256AC-10L, RAM
U4	1	74HC373, LATCH
U5	1	74HC00, NAND
U6	1	SN75176B, BUS XCEIVER
U7	1	MAX233CPP, RS-232 DRIVER
U8	1	74HCT245, LINE DRIVER
U9	1	74HCT244, LINE DRIVER
C1,C2	2	33pF, 7V, (ceramic)
C3	1	4.7uF, 25V, (elec.)
C4	1	0.1uF, 16V, (tant.)
C5,C11	2	10uF, 16V, (elec.)
C6,C7,C8,C9, C10,C12,C13	7	0.1uF, 7V, (ceramic)
C14	1	.01uF, 16V, (tant.)
R1,R2	2	200 Ohm
R3	1	8.2K Ohm
R4	1	4.7K Ohm
R5,R6	2	86 Ohm
R7	1	23 Ohm
R8	1	4.7 Ohm
D1,D2	2	LED, 2.3v Green, PC Mount, 550 Series
D4	1	Zener Diode, 5.1V, 0.25W
D5,D6,D7,D8	4	Zener Diode, 10V, 0.25W

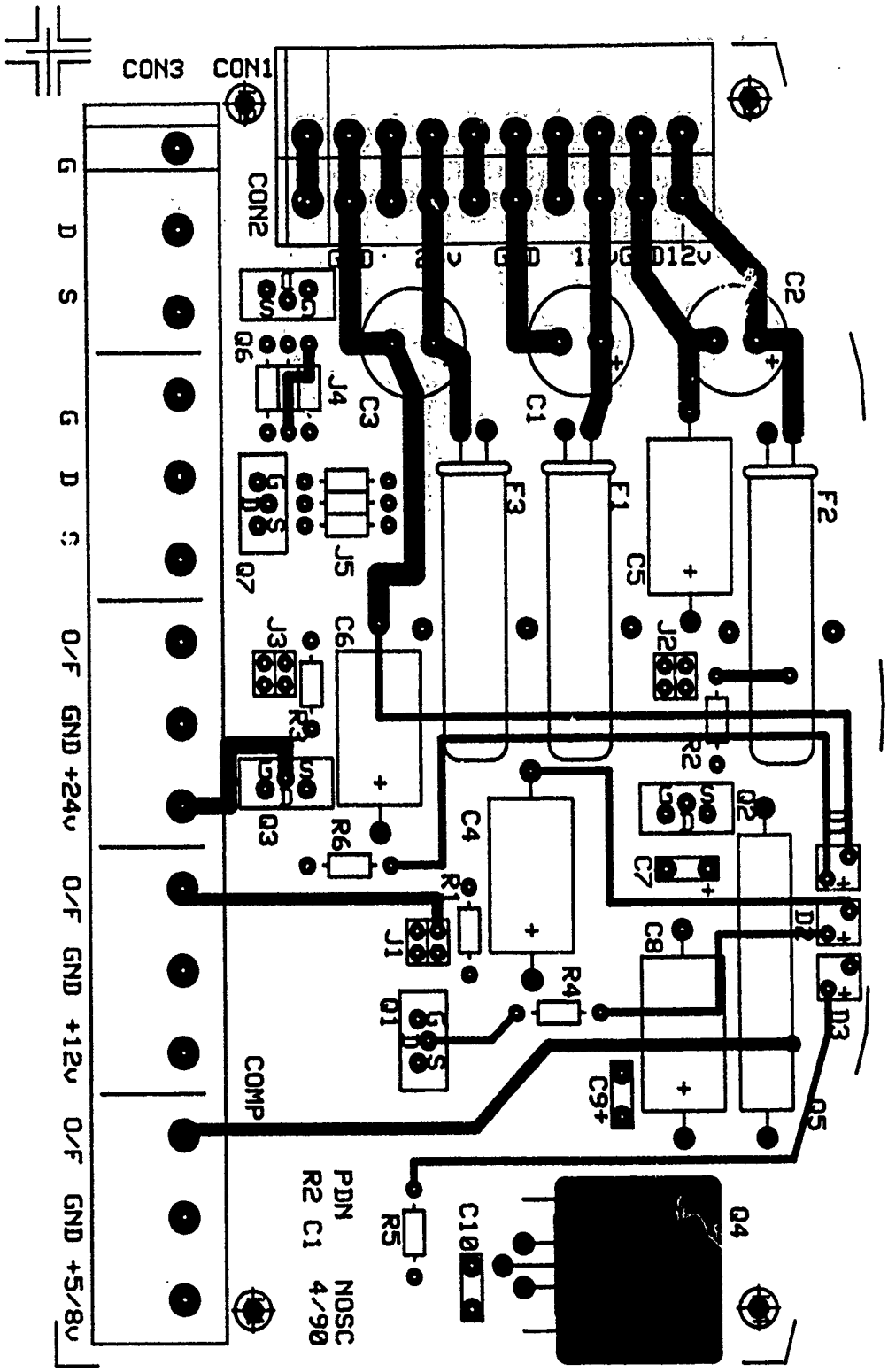
Board Ref.	Quan.	P/N - Discription
Q1	1	Voltage Regulator, 5V, 1A
Q2	1	Crystal Osc., 14.7456 MHz
SW1,SW2	2	Dip Switch, 8-pin, PC Mount, SPST
SW3	1	Momentary Push Button, PC Mount, NO
J1	1	Header, 8-pin, PC Mount, Vertical
CONN 1	1	DB-25 Male, PC Mount, Right Angle
CONN 2	1	2 Pin Screw Terminal, PC Mount
CONN 3	1	Phone Jack, 6-pin, PC Mount, Right Angle
CONN 4	1	Phone Jack, 6-pin, PC Mount, Vertical

file: icnparts.doc
last revised: 6/25/91

PDN SCH



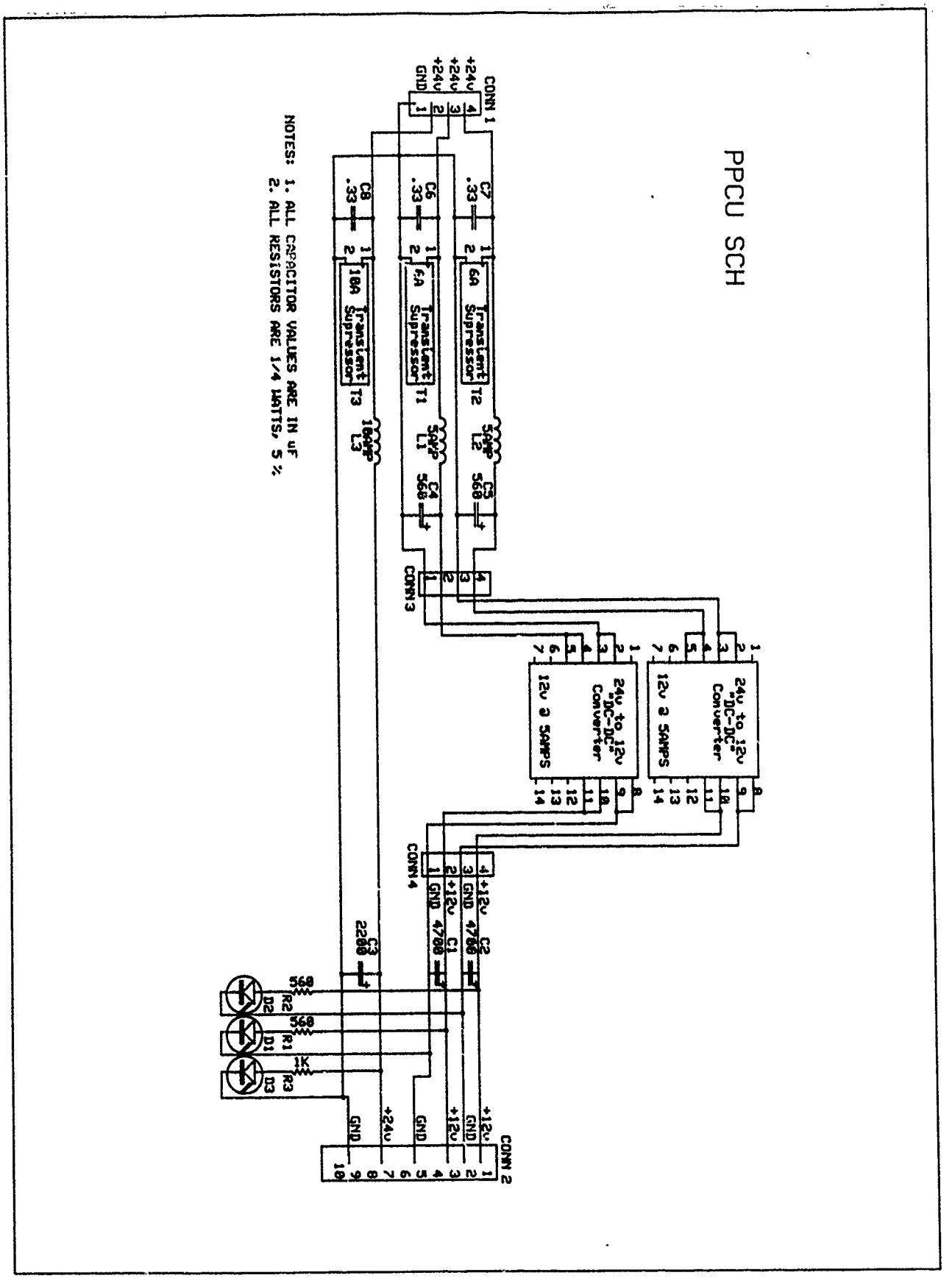
NOTES: 1. ALL CAPACITOR VALUES ARE IN uF
2. ALL RESISTORS ARE 1/4 WATT, 5 %
3. FROM PPCU CONN 2 or PREVIOUS PDN CONN 2
4. J1,J2,J3 B-JUMPER TURNS POWER ON AT CONN 3
J1,J2,J3 A-JUMPER TURNS POWER OFF AT CONN 3 AND VOLTAGE BECOMES SWITCHABLE VIA. THE O/F PIN ON CONN 3
5. F1,F2,F3 ARE AUTO-RESET THERMAL BREAKERS, VALUES SHOWN ARE MAX.

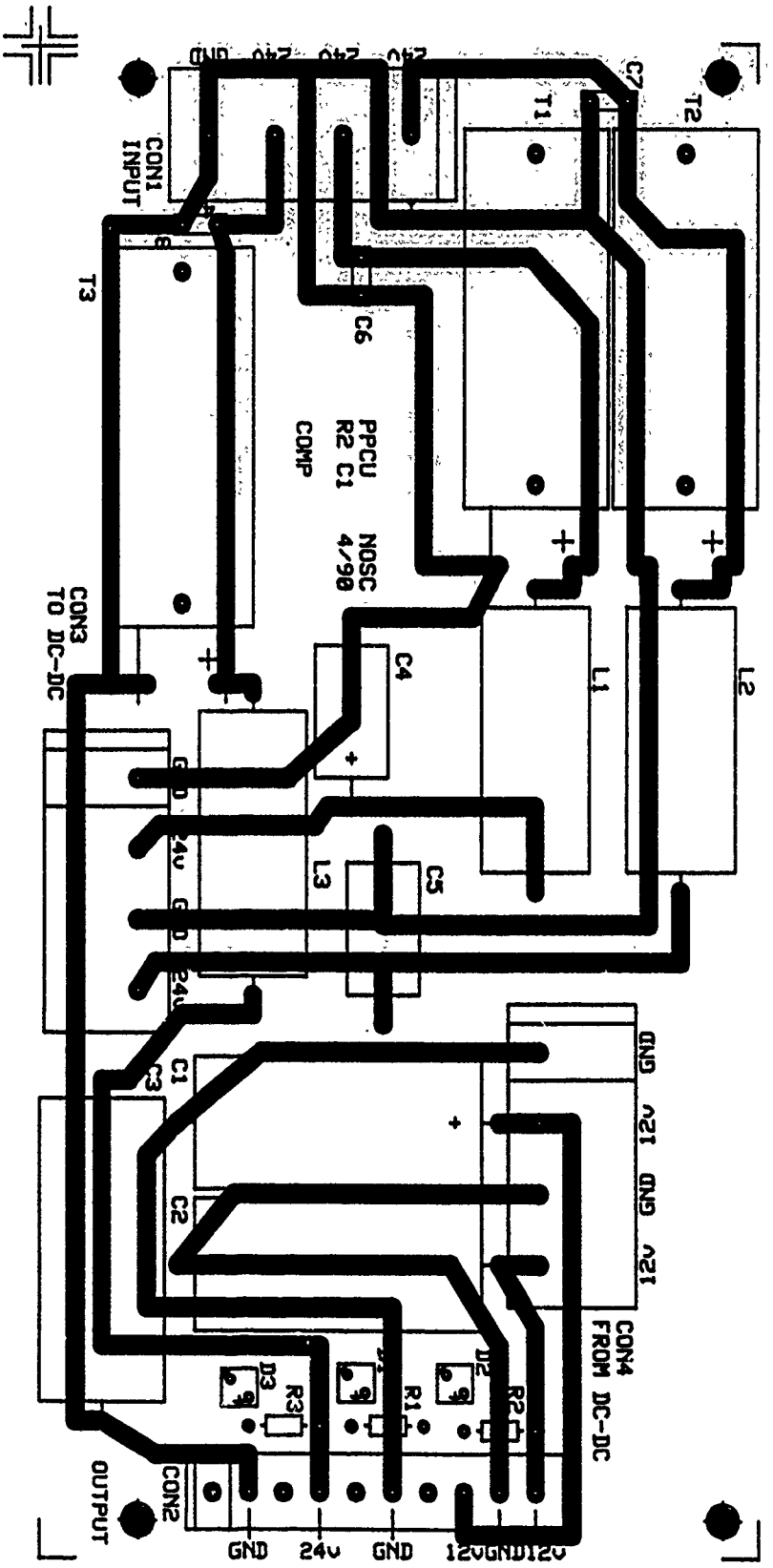

CON3
CON1
CON2
G D S G D S
O/F GND +24v
O/F GND +12v
O/F GND +5/8v
COMP
PDN
R2 C1
NOSC
4/90
Q1
Q2
Q3
Q4
Q5
Q6
Q7
J1
J2
J3
J4
J5
C1
C2
C3
C4
C5
C6
C7
C8
C9
C10
R1
R2
R3
R4
R5
R6
F1
F2
F3
D1
D2
D3

Power Distribution Node (PDN) Parts List

Board Ref	Quan.	P/N - Discription
C3	1	470uF (elec.),35V
C1,C2	2	470uF (elec.),25V
C4,C5,C6,C8	4	560uF (elec./tant.),25V
C7,C9	2	0.33uF (tant.)
C10	1	0.1uF (tant.)
R1,R2,R3	3	10K Ohm
R4,R5	2	560 Ohm
R6	1	1K Ohm
D1,D2,D3	3	LED, 2.3V Green, PC Mount, 550 Series
J1,J2,J3	3	2x2 PC Mount Jumpers
F1,F2	2	3A Resetable Circuit Breaker
F3	1	6A Resetable Circuit Breaker
Q1,Q2,Q3,Q6, Q7	5	IRF 9531, P-Channel MOSFET's, TO-220 style
Q4	1	7805/7808 Voltage Regulator
CONN 2	1	10 pin Vertical Terminal Strip, 8213 Series
CONN 1	1	10 pin Horizontal Terminal Strips, 8213 Series
CONN 3	1	15 Conductor Screw Terminal

file: pdnparts.doc
last revised: 6/10/90


PPCU SCH
NOTES: 1. ALL CAPACITOR VALUES ARE IN uF
2. ALL RESISTORS ARE 1/4 WATTS, 5 %
CONN 1
+24v
+24v
+24v
GND
C7 .33
C6 .33
C8 .33
6A Transient Supressor T2
6A Transient Supressor T1
10A Transient Supressor T3
5AMP L2
5AMP L1
10AMP L3
C3 560
C4 560
CONN3
24v to 12v "DC-DC" Converter
12v @ 5AMPS
24v to 12v "DC-DC" Converter
12v @ 5AMPS
CONN4
+12v
GND
GND
+12v
C2 4700
C1 4700
C3 2200
560 R2
560 R1
1K R3
D2
D1
D3
GND
+24v
GND
+12v
GND
+12v
CONN 2


CON1
INPUT
T1
T2
T3
C7
C6
PPCU
R2 C1
COMP
NOSC
4/98
C4
L1
L2
CON3
TO DC-DC
L3
C5
C1
C3
C2
GND
12V
GND
12V
CON4
FROM DC-DC
D3
R3
R1
D2
R2
OUTPUT
CON2
GND
24V
GND
12V
GND
12V

Platform Power Conditioning Unit (PPCU) Parts List

Board Ref.	Quan.	P/N - Discription
C1,C2	2	4700uF (elec.)
C3	1	2200uF (elec.)
C4,C5	2	560 uF (tant.)
C6,C7,C8	3	0.33 uF (tant.)
R1,R2	2	560 Ohm
R3	1	1K Ohm
L1,L2	2	5A Current Choke, 5200 Series
L3	1	10A Current Choke, 5200 Series
T1,T2,T3	3	15A, 28V Transient Suppressor
D1,D2,D3	3	LED, 2.3V Green, PC Mount, 550 Series
CONN 2	1	10 Conductor Pluggable Terminal Strip, PC Mount, 8213 Series
CONN 1,3,4	3	4 Conductor Screw Terminal Strips, PC Mount
	2	24-12V Dc-DC Converter, WR24S12/60K3

file: ppcuparts.doc
last revised: 6/12/91

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 1991	Final

4. TITLE AND SUBTITLE
DEVELOPMENT OF A MODULAR ROBOTIC ARCHITECTURE

5. FUNDING NUMBERS
PE: 0602936N
WU: DN300029

6. AUTHOR(S)
R. T. Laird, R. P. Smurlo, and S. R. Timmer

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Ocean Systems Center
San Diego, CA 92152-5000

8. PERFORMING ORGANIZATION REPORT NUMBER
NOSC TD 2171

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Office of Chief of Naval Research
Arlington, VA 22217

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

This objective of this project is to develop the hardware and software components for constructing and controlling reconfigurable, modular robots.

14. SUBJECT TERMS
modular robots
mobile security robots
independent modules
control station
remote platform

15. NUMBER OF PAGES
200

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAME AS REPORT

21a. NAME OF RESPONSIBLE INDIVIDUAL	21b. TELEPHONE (Include Area Code)	21c. OFFICE SYMBOL
R. T. Laird	(619) 553-3667	Code 535

INITIAL DISTRIBUTION

Code 0012	Patent Counsel	(1)
Code 0144	R. November	(1)
Code 535	R. T. Laird	(15)
Code 952B	J. Puleo	(1)
Code 961	Archive/Stock	(6)
Code 964B	Library	(3)

Defense Technical Information Center
Alexandria, VA 22304-6145 (4)

NCCOSC Washington Liaison Office
Washington, DC 20363-5100

Center for Naval Analyses
Alexandria, VA 22302-0268

Navy Acquisition, Research & Development
Information Center (NARDIC)
Alexandria, VA 22333

Navy Acquisition, Research & Development
Information Center (NARDIC)
Pasadena, CA 91106-3955